AFRL-VA-WP-TR-2004-3105

HIERARCHICAL CONTROL OF SEMI-AUTONOMOUS TEAMS UNDER UNCERTAINTY (HICST)

Pravin Varaiya

University of California
Department of Electrical Engineering and Computer Sciences
Berkeley, CA 94720

MAY 2004

Final Report for 28 September 2001 – 31 December 2003

Approved for public release; distribution is unlimited.

STINFO FINAL REPORT

AIR VEHICLES DIRECTORATE
AIR FORCE MATERIEL COMMAND
AIR FORCE RESEARCH LABORATORY
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE U.S. GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT HAS BEEN REVIEWED BY THE AIR FORCE RESEARCH LABORATORY WRIGHT SITE OFFICE OF PUBLIC AFFAIRS (AFRL/WS/PA) AND IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONALS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

/s/

MARK J. MEARS
Project Engineer
Control Design and Analysis Branch

/s/

HOWARD T. EMSLEY, *Actg Chief*
Control Design and Analysis Branch
Control Sciences Division

/s/

BRIAN W. VAN VLIET, Chief
Control Sciences Division
Air Vehicles Directorate

COPIES OF THIS REPORT SHOULD NOT BE RETURNED UNLESS RETURN IS REQUIRED BY SECURITY CONSIDERATIONS, CONTRACTUAL OBLIGATIONS, OR NOTICE ON A SPECIFIC DOCUMENT.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2004	Final	09/28/2001 – 12/31/2003

4. TITLE AND SUBTITLE
HIERARCHICAL CONTROL OF SEMI-AUTONOMOUS TEAMS UNDER UNCERTAINTY (HICST)

5a. CONTRACT NUMBER
F33615-01-C-3150

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0602301

6. AUTHOR(S)
Pravin Varaiya

5d. PROJECT NUMBER
A055

5e. TASK NUMBER

5f. WORK UNIT NUMBER
0A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California
Department of Electrical Engineering and Computer Sciences
Berkeley, CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Vehicles Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, OH 45433-7542

Defense Advanced Research Projects Agency (DARPA)
3701 N. Fairfax Avenue
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/VACA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-VA-WP-TR-2004-3105

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Report contains color.

14. ABSTRACT

This is the final report of work done under DARPA Contract F33615-01-C-3150, for the period of performance September 2001 through December 2003. Algorithms and associated software were developed for the following modules: 1. Interactive task planner (ITP); 2. Configuration and schedule; 3. Task execution; 4. State estimator; 5. Java interface to OEP; 6. Robust dynamic programming for path planning with uncertain information; 7. Flexible formation of teams operating under large uncertainties; and 8. Path planning with two constraints. Modules 1-5 are integrated into a self-contained package that can be used in an off-line or open loop planning phase followed by a closed-loop execution phase. The package can be used in a fully automated fashion or in an interactive manner, in which the user can intervene at several stages to modify the operation of the modules. Thus the package makes provision for 'mixed initiative'. Modules 6-8 are 'stand alone' algorithms. Software implementations for these algorithms were developed. The report describes these modules and provides examples to illustrate their operation. The technology developed under HICST, and the modules that embody this technology, represent a significant advance towards the objectives of the MICA program.

15. SUBJECT TERMS
Cooperative Control, Unmanned Air Vehicles

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mark J. Mears
Unclassified	Unclassified	Unclassified	SAR	152	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-8685

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Contents

1 Introduction **7**

2 Summary **10**

2.1 Module 1: Interactive task planner 10

2.2 Module 2: Configuration and schedule 13

2.3 Module 3: Task execution 17

2.4 Module 4: Database 21

2.5 Module 5: Java 21

2.6 Module 6: Robust path planning 23

2.7 Module 7: Dynamic team formation 24

2.8 Module 8: Path planning with two constraints 26

3 Module 1: Interactive task planner **28**

3.1 Threat 28

3.2 Plan design space 30

3.3 Risk along a path 31

3.4 Value function calculation 33

3.5 Risk of a plan with prespecified order of attack 34

3.6 The ITP procedure and optimal plan 34

3.7 Refinement of the ITP procedure: mixed initiative 36

4 Module 2: Configuration and schedule **38**

4.1 Task scheduling in the MICA context 38

4.2 Scheduling model formulation and solutions 40

4.3 Team composition 42

4.4 Linear programming formulation 43

4.5 Goal programming formulation 44

4.6 Implementations in executable code 45

5 Module 3: Task execution **48**

5.1 Introduction . . . 48

5.2 An aside on *Shift* . . . 49

5.3 Architecture . . . 50

5.4 Mixed initiative interactions . . . 53

5.5 UCAV type . . . 53

5.6 Platform type . . . 53

5.7 Maneuver specification . . . 55

5.7.1 Base type . . . 55

5.7.2 Types of maneuvers . . . 55

5.7.3 Example: attack_jam . . . 55

5.8 Maneuver controller . . . 56

5.8.1 Base type . . . 56

5.8.2 Example: attack_jam type . . . 57

5.9 Vehicle supervisor . . . 57

5.10 Vehicle dispatcher . . . 58

5.10.1 Mission specification . . . 58

5.10.2 Dispatcher type . . . 59

5.11 Task specification . . . 59

5.11.1 Concepts . . . 59

5.11.2 Leg type . . . 60

5.11.3 Subtask type . . . 61

5.12 Team controller . . . 61

5.12.1 Base type . . . 61

5.12.2 Task controller . . . 61

5.12.3 Sub-task controller . . . 62

5.12.4 Properties . . . 66

5.13 Conclusion . . . 66

6 Module 4: State estimator **72**

6.1 Threat distribution after strike . . . 72

6.2 Threat distribution after search . . . 74

6.3 Implementation . . . 75

7 Module 5: Java interface to OEP **76**

7.1 Java client to the OEP . . . 76

7.2 RMI Services . . . 76

8 Module 6: Robust path planning **78**

8.1 Introduction . . . 78

8.2 Problem Setup . . . 79

8.2.1 The Bellman recursion . . . 79

8.2.2 Addressing uncertainty in the transition matrices . . . 80

8.2.3 The robust Bellman recursion . . . 81

8.2.4 Main result . . . 81

8.3 Robust algorithm summary . . . 82

8.4 Likelihood Models . . . 82

8.4.1 Model description . . . 83

8.4.2 The dual problem . . . 84

8.4.3 A bisection algorithm . . . 85

8.5 Maximum a posteriori models . . . 86

8.6 Entropy Models . . . 87

8.6.1 Model description . . . 87

8.6.2 Dual problem . . . 87

8.6.3 A bisection algorithm . . . 88

8.7 Other Specific Models . . . 89

8.8 Interval matrix model . . . 89

8.9 Ellipsoidal models . . . 89

8.10 Example: Robust Aircraft Routing . . . 91
8.11 The nominal problem . . . 91
8.12 The robust version . . . 92
8.13 Comparing robust and nominal strategies . . . 92
8.14 Inaccuracy of uncertainty level . . . 93
8.15 Concluding remarks . . . 94
8.16 Appendix . . . 95
8.16.1 Proof of the robust Bellman recursion . . . 95
8.16.2 Properties of function ϕ of section 8.4.3 . . . 96
8.16.3 Properties of function ϕ of section 8.6.3 . . . 97
8.16.4 Calculation of β for a Desired Confidence Level . . . 98

9 Module 7: Flexible team formation **103**
9.1 Problem Statement . . . 103
9.2 Constraints and optimization objective . . . 104
9.3 Multiple resources allocation . . . 106
9.4 Dealing with integer approximations . . . 107
9.5 Resource Allocation under Uncertainty . . . 108
9.6 Scenario-based optimization . . . 109
9.7 Approximate feasibility of scenario solutions . . . 110
9.8 A posteriori analysis . . . 111
9.9 Interaction models . . . 112
9.10 Numerical examples . . . 112
9.11 The nominal problem . . . 113
9.12 The Robust counterpart . . . 114
9.13 Conclusion . . . 115

10 Module 8: Path planning with multiple constraints **118**
10.1 Introduction . . . 118

10.1.1 Problem definition . . . 119
10.1.2 Related work . . . 119
10.2 Value function solution . . . 120
10.2.1 Single objective shortest path . . . 120
10.2.2 Computing path integrals . . . 121
10.2.3 Exploring potential paths . . . 122
10.2.4 Numerical algorithms . . . 123
10.3 Examples . . . 124
10.3.1 Two costs in two dimensions . . . 125
10.3.2 Three costs in two dimensions . . . 126
10.3.3 Two costs in three dimensions . . . 127
10.3.4 The implementation and deExecution times . . . 127
10.4 Discussion . . . 129
10.5 Appendix: Update equations for any number of dimensions . . . 130

11 Conclusions **137**
11.1 Planning . . . 137
11.2 Execution . . . 138
11.3 State estimation . . . 138
11.4 Re-planning . . . 139

12 Open problems **140**
12.1 Planning . . . 140
12.2 State estimation . . . 141
12.3 Execution . . . 141
12.4 Re-planning . . . 142

References **143**

1 Introduction

This is the final report of work done in the project, "Hierarchical control of semi-autonomous teams under uncertainty (HICST)," under Darpa Contract F33615-01-C-3150. The period of performance was September, 2001 to December, 2003. The contract was awarded by the MICA (Mixed Initiative Control of Automa-teams) Program. MICA's objectives are to:

- Develop theory, algorithms, software, and modeling/simulation capabilities for hierarchical battlespace management and distributed control of semi-autonomous entities
  - Cultivate dynamic operational and mission planning for teamed entities
  - Develop cooperative path/execution planning
  - Address an active, intelligent adversary and threats in an uncertain environment
- Demonstrate multiple vehicle execution of team-based strategies

These objectives become clearer in the following setting.[1] There is a set of Unmanned Air Vehicles (UAVs) with different capabilities (sensors, weapons, decoys), collectively called the *Blue force*, and a set of targets called the *Red force*. The Blue force is used to attack the Red force, which threatens the attacking Blue force. The HICST project developed some of the theory, algorithms, and associated software for planning and executing Blue's attack.

A *plan* organizes Blue's attack into a set of independent *tasks*.[2] Each task is a list of *sub-tasks*. Each sub-task comprises a list of targets to be attacked in a specific order. A *team* of several UAVs is needed to carry out each sub-task, so a plan must also assign a team of UAVs to each sub-task. The specification of a team includes the UAV platforms (size, speed) and their configuration, including the weapons and sensors each UAV carries. The plan also specifies the *dependencies* among the sub-tasks: This is a partial order or *precedence* relation that says that certain targets must be attacked before others. The plan may also impose timing constraints on sub-task completion. Lastly, the plan determines a nominal path for the UAVs in each team. The nominal path for a team is later refined to specify a *mission* for each UAV in the team. In summary, a plan for a single task comprises:

1. The decomposition of Blue's attack into tasks, each of which is a list of subtasks;
2. A team for each task comprising a set of configured UAVs;
3. A precedence relation among sub-tasks and timing constraints;
4. A nominal flight path for each team.

A plan is designed on the basis of prior information, called the Intelligent Preparation of the Battlefield (IPB). The plan is sometimes called a Team Composition and Tasking (TCT) plan for 'team composition and tasking'.

[1] A concrete instance of this setting is the Open Experimental Platform or OEP, developed by Boeing under the MICA program. The modules described below are intended for the OEP.

[2] 'Independent' tasks can be executed without any timing or precedence order among them.

The real-time *execution* of a plan is governed by a two-layer controller, whose upper layer determines team coordination, and whose lower layer determines the control of individual UAVs in a team. The control architecture is thus organized in the three layers of figure 1. The TCT Plan layer is 'offline' and precedes the online execution of the attack.

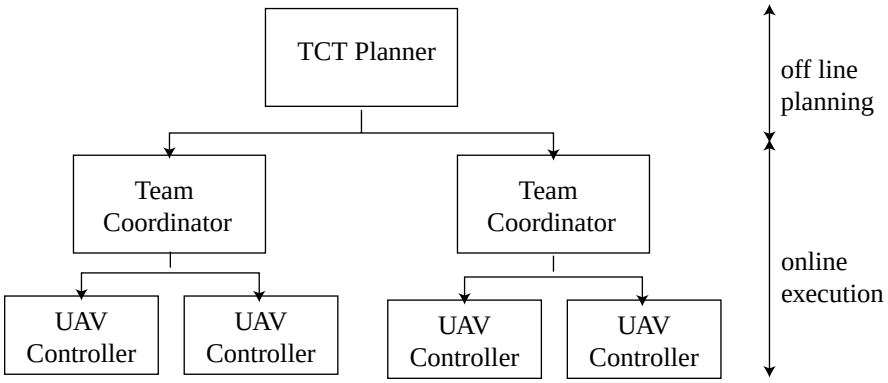


Figure 1: The TCT Planner designs the plan, which is executed by a two-layer controller that coordinates team effort and controls the individual UAVs.

Figure 1 suggests that once the TCT plan is designed it is 'handed' over for execution. However, this may not be the case. It may happen that, during execution, new information is received that triggers a re-working of the plan. This introduces the 'feedback' loop indicated in figure 2.

The planning and execution phases may be fully automated. However, the control design permits human intervention in both phases to guide or override the automated choices. Thus there is provision for 'variable' autonomy.

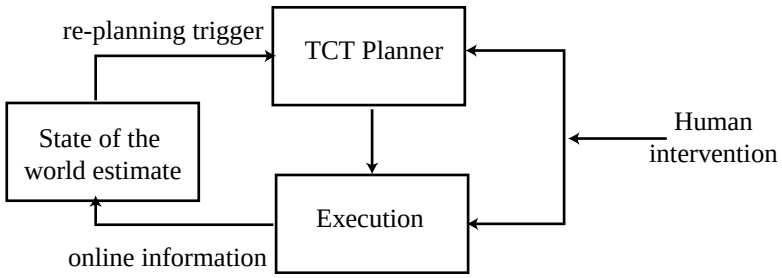


Figure 2: Information received during plan execution leads to a change in the estimate of the 'state of the world' and may trigger a re-working the plan, creating a feedback loop.

The work conducted under the HICST project is conveniently summarized as algorithms and associated software, organized in the following modules:

1. Interactive task planner (ITP);
2. Configuration and schedule;

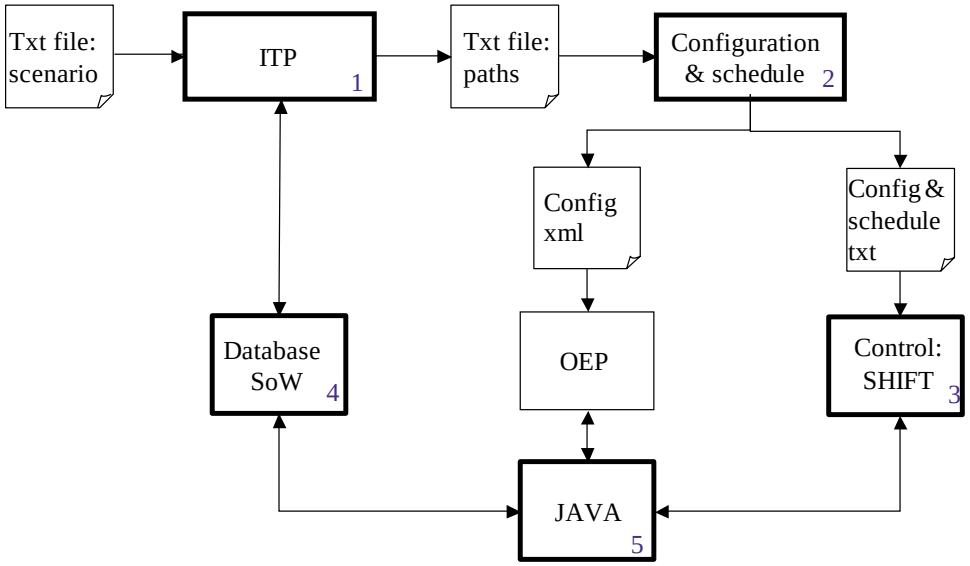


Figure 3: Integration of modules 1-5. The modules make provision for human intervention, not indicated in the figure. SoW is 'state of the world'.

3. Task execution;
4. Database for state estimation;
5. Java interface to OEP;
6. Robust dynamic programming for path planning with uncertain information;
7. Dynamic team formation operating under large uncertainties;
8. Path planning with two constraints.

Modules 1-5 are integrated into a self-contained package as indicated in figure 3, in which the five modules are labeled and encapsulated in bold boxes. The package may be operated in an open-loop or 'planning' mode or in a closed loop or 'execution' mode. Modules 1-5 make provision for human intervention at well-defined junctures. This is not indicated in figure 3, but is discussed in later sections.

The remainder of the report is organized as follows. Section 2 briefly describes each of the eight modules. The interested reader can learn about the underlying theory and algorithm for each module in the following eight sections. Section 11 summarizes our main contribution, and section 12 gives our perspective on the difficult open problems.

2 Summary

We briefly describe each module; in each case we specify the input, output, and what the module does.

2.1 Module 1: Interactive task planner

The ITP module takes as input a txt file, which specifies the SW and NE corners of the scenario area, the location of the blue base, and the location, type, and range of the red threats. Figure 4 is an excerpt of this input file.

$BASE	-83.47249703	-244.9023235			
$SW	-445.0066235	-445.278318			
$NE	445.010847	445.2816576			
#name	oepid	oeptype	location		range
ew1	ew1	ew_radar_site_type	83.08835062	357.8026933	0
ew2	ew2	ew_radar_site_type	84.52129045	257.5534701	0
ew3	ew3	ew_radar_site_type	105.5589836	136.3371254	0
ew4	ew4	ew_radar_site_type	217.4891654	152.4172989	0
ew5	ew5	ew_radar_site_type	313.5345099	140.1720993	0
ew6	ew6	ew_radar_site_type	314.9126829	229.1353302	0
c2_1	c2_1	c2_facility_type 143.4283105	196.6354571	0	
c2_2	c2_2	c2_facility_type 237.343399	196.6354571	0	
long_sam1	long_sam1	long_sam _fire_control_platform_type	-15.94678417	377.0053928	80
long_sam2	long_sam2	long_sam _fire_control_platform_type	83.87181924	309.8998433	80
long_sam3	long_sam3	long_sam _fire_control_platform_type	94.1244268	184.7984307	80
long_sam4	long_sam4	long_sam _fire_control_platform_type	176.454496	137.30932	80
long_sam5	long_sam5	long_sam _fire_control_platform_type	280.0729363	135.7397081	80

Figure 4: Excerpt of ITP input file, specifying SW and NE corners of scenario area, location of blue base, and red threats.

After receiving this input file in *Step 1*, the ITP creates a scenario window that displays the location of the threats and the blue base. In *Step 2*, the planner selects a subset of the threats as *primary*. These are the threats that the planner wants to attack. The ITP then calculates and displays additional, *potential* threats. These are the threats that 'protect' the primary threats, i.e. one or more primary threats are included in the threat range of the potential threats. The planner next chooses a *minimum* risk level.

In *Step 3*, the ITP determines the subset of potential and primary threats that can be attacked along paths starting at the blue base, for which the total risk is less than the chosen minimum risk level. This subset of targets is called *Wave 1*. The ITP also determines the nominal risk-minimizing paths from the blue base to each of Wave 1 targets.

These paths are displayed in the scenario window. They are called 'nominal' because the actual paths taken by the UAV are somewhat different and are determined by the 'task execution' module.

In *Step 4*, the ITP determines the subset *Wave 2* of additional targets that can be attacked along paths starting at either the blue base or one of the Wave 1 target locations, for which the total risk is less than the minimum. The ITP determines and displays the nominal risk-minimizing paths for Wave 2 targets. The ITP also displays the optimal 'minimum risk' contours of locations that can be reached for each level of total risk.

The procedure continues in this way, generating Wave 3, Wave 4, etc. until all the primary targets have been reached.

The elimination of a target m in Wave i reduces the risk for each target in Wave $i+1$ by, say, the amount $S(m,n)$. S is called the sensitivity matrix. The planner may remove target m^* from Wave i if the sensitivies $S(m^*,n)$ are small.

On the other hand, before the calculation of any wave, the planner may add new threats to a wave, and change the minimum acceptable risk.

Figure 5 is a snapshot of the ITP display window. 'Wave threshold' is the minimum risk level the planner selects; 'value' is the minimum risk incurred by the various paths. The 'snapshot' window in the top right allows the planner to revisit earlier decisions and change them, if necessary.

The output of ITP thus consists of:

- A set of targets, organized in waves, and the minimum risk for each target, based on the assumption that targets in Wave $i+1$ are attacked after those in Wave i have been destroyed;
- A nominal risk-minimizing path for each target in Wave $i+1$, starting at the blue base or at any target locations in Wave 1-i;
- A 'sensitivity' matrix that gives the reduction in risk for each target in Wave $i+1$ due to the elimination of each target in Wave i.

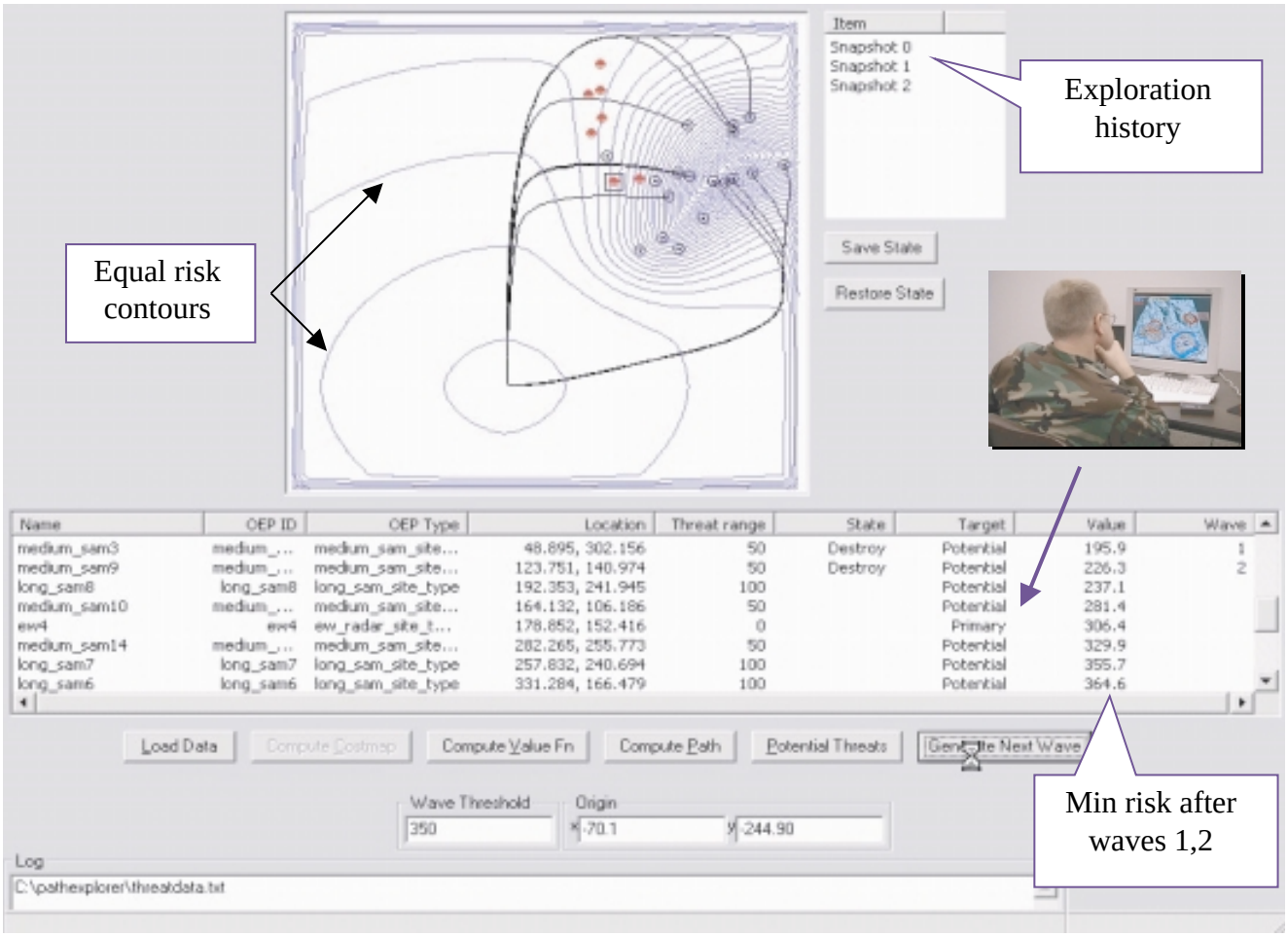


Figure 5: The ITP display window gives the paths, the Wave targets, the risk incurred along each path, and the equal risk contours.

2.2 Module 2: Configuration and schedule

This module takes the ITP output as its input and produces the two files indicated in figure 3. We explain how this is done. The top diagram of figure 6 represents in graphical form part of the information in the ITP output. There are four final targets. The ITP organizes Blue's attack in three waves, with four targets each. For reference, we index the 12 targets by the pair (i, Wave j) to mean the ith target in Wave j. The arrows in the diagram refer to the origin and destination of the minimum risk paths generated by the ITP. Thus, for example, the minimum risk path to target (4, Wave 2) starts at the blue base. Similarly, the minimum risk path to target (3, Wave 3) starts at (4, Wave 2). The ITP output also includes the sensitivity matrix described at the end of section 2.1 above.

The 'configuration and schedule' module produces information depicted in the middle and bottom diagrams. It organizes some of the targets into sub-tasks. In this case, there are four sub-tasks. For example, sub-task 1 consists of targets (2, Wave 1), (1, Wave 2) and (1, Wave 3). All final targets are included in the sub-tasks. However, three targets, (1, Wave 1), (4, Wave 1), and (2, Wave 2) are not included in any sub-task. Presumably, the module determined that, based on the sensitivity matrix, these targets do not contribute to a significant reduction in risk, so they are excluded.

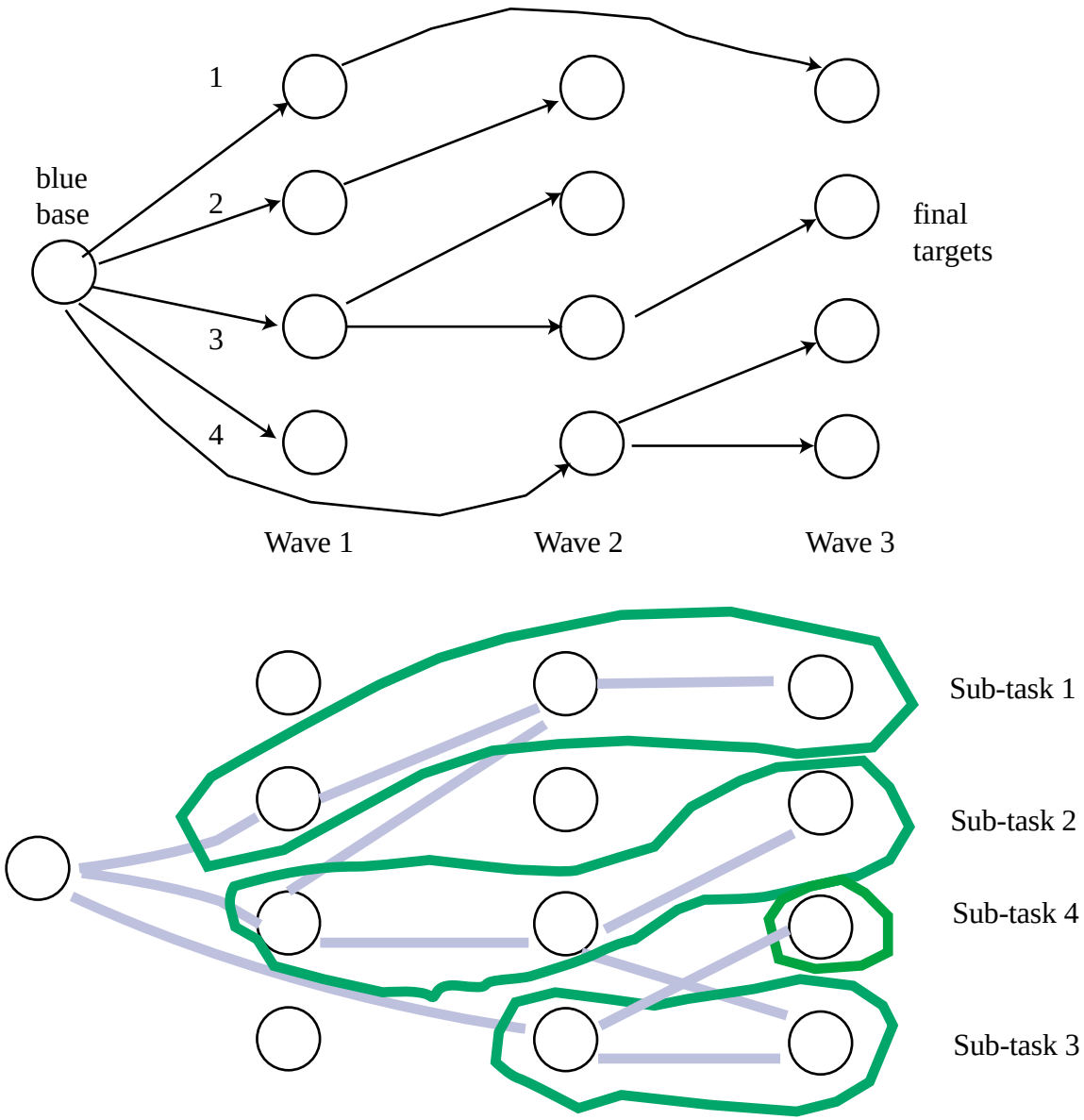
The solid lines in the middle diagram indicate the precedence relation. Thus target (1, Wave 2) can only be attacked after targets (2, Wave 1) and (3, Wave 1) have been destroyed. Observe that this precedence relation is *weaker* than that implicit in the ITP calculation, which suggests that all targets in Wave j must be destroyed before any target in Wave $j+1$ is attacked. The weaker precedence relation takes into account the sensitivities, which are not considered in the ITP's automatic wave calculation. From a planning perspective, the weaker relation is desirable because it imposes fewer dependencies among subtasks, reducing coordination among teams and increasing flexibility in timing.

Having determined the task composition and the precedence relation, this module determines the team configuration for each task. Consider task 1, which includes three subtasks, the targets (2, Wave 1), (1, Wave 2) and (1, Wave 3). The 'configuration and schedule' module takes into account the threats at these targets (long range SAM, medium range SAM, etc.), determines the weapons to be used to attack each target and their lethality, and specifies Team 1, i.e. the platforms and their configuration (weapons, sensors). The table at the bottom of the figure gives an example of a Team specification this module produces.

The two files output by this module indicated in figure 3 are textual representations of the information in figure 6. The xml file describes the team configurations in a form that the OEP accepts and is used to instantiate the teams. The 'config & schedule' txt file describes the sub-tasks for each team for the 'Task execution' module.

The final output of this module is the determination of the paths and precedence relationships to be followed by each UAV. (This is the UAV's mission.) Recall that the ITP produces a nominal path for a *team* for each sub-task. The module modifies this nominal path and specifies a path that each UAV in the team must follow during the execution phase. We explain how this is done with the help of figure 7.

The figure shows two scenarios. The initial location of the three-UAV team is at A and the target is at B, in both cases. In the upper scenario, there is only one threat, indicated by the circle around B. In the lower scenario, there an additional threat, indicated by the circle around C. The ITP gives the nominal team path as the straight line starting at A and ending at the target at B for both scenarios. Suppose that the module has determined that the target at B is



Task 2 - Team Bravo

Target Scenario Name	Weapon Selection	Number of Weapons on Target	Probability of Destruction	Release distance(km) at 5 kft, and 153 knots	Platform
ew3	GPS Bomb	2	2 drawings of 0.95	5km	Wolf_1
ew4	GPS Bomb	2	2 drawings of 0.95	5km	Coyote_1
ew5	GPS Bomb	2	2 drawings of 0.95	5km	Coyote_2

Figure 6: The ITP output (top) is analyzed to produce sub-tasks and precdence relation (middle), and team composition (bottom).

to be attacked by a single UAV (called the attack UAV in the figure); the two remaining UAVs are ‘reserve’ UAVs to be used for later targets in the sub-task.

There is an operational constraint: to successfully attack its target, the UAV heading must be in the direction of the target in order to minimize its signature; if the UAV is not headed in this direction, its signature is larger, increasing the likelihood of its detection by radar and consequent destruction.

Consider now the upper scenario. If the attack UAV follows the nominal path it will meet the operational constraint. The nominal path is decomposed into two ‘legs’: leg 1 which terminates at the threat region is safe; leg 2 is ‘unsafe’ but meets the operational constraint. So the attack UAV can proceed from leg 1 to leg 2 and destroy the target at B. The reserve UAVs can advance along leg 1, go into a holding pattern at the end of leg 1, and proceed along leg 2 only after the target at B is destroyed.

Consider now the lower scenario. If the attack UAV follows the nominal path, it will exhibit a larger signature to the radar at C and it will get destroyed. So the nominal path must be modified as indicated. If the attack UAV follows leg 2, it will minimize its signature to the radars at both B and C; it can then destroy the target at B, proceed to destroy the target at C. The reserve UAVs can then proceed after these targets are destroyed.

This example shows how this module modifies the nominal path to minimize the signature of the attack UAV, decomposes it into legs, and introduces precendence or dependency relations among the legs of the sub-tasks. This precedence relation among legs is a refinement of the precendece relation among sub-tasks.

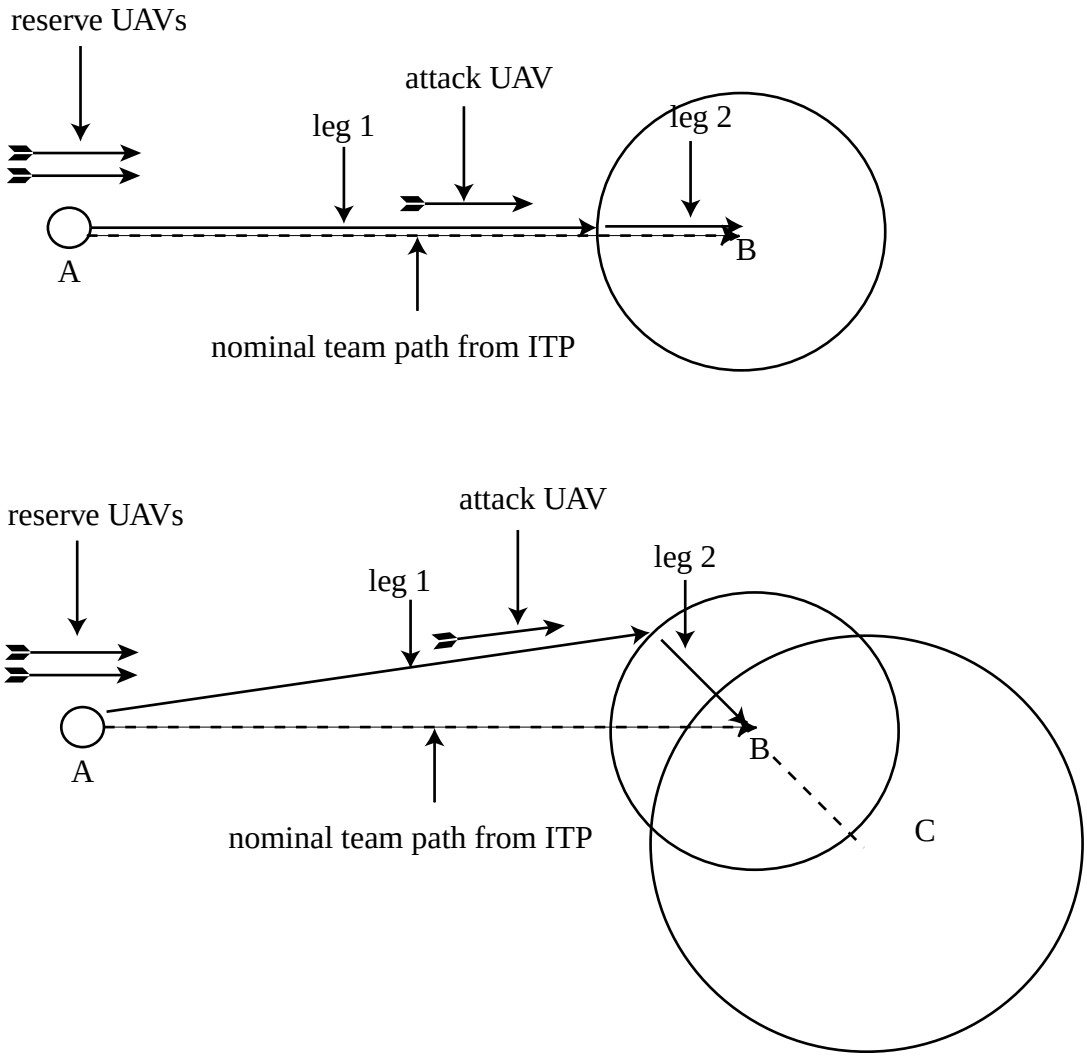


Figure 7: A team's nominal path from A to the target at B is modified into a sequence of two legs, the first of which is 'safe', and the second of which is followed in the 'attack' phase. The UAVs that are not part of the attack are held in 'reserve'. When the target is 'protected' by another threat (at C), the path is modified to minimize the signature of the attack UAV to both B and C.

2.3 Module 3: Task execution

The 'task execution' module is a Shift[3] program, which takes as an input a txt configuration and schedule file (the specifications for the execution phase, which are output by module 2) and executes the specification in the OEP simulation environment. The Shift program consists of a hierarchy of controllers for the teams and the individual UAVs.

The execution module is coupled to the OEP through the JAVA module, indicated in figure 3, through which the controllers receive sensor observations and send control commands for way-points, sensors, and weapons.

The input txt file specifies: sub-tasks; legs for each sub-task; precedence relations on sub-task legs; and teams assigned to execute each sub-task. The txt file *paths.hs* is encoded in Shift. Figure 8 is an excerpt of this input file.

The user interactions take place within the Microsoft Visual C++ environment. In *Step 1*, the user runs the OEP batch files to start the OEP: *naming_service, c4isim_server, load_scenario, openmap*.

In *Step 2*, the user starts Microsoft Visual C++ and loads the Shift workspace. The Shift workspace includes the file *shift_oep.hs*, as well as the Shift engine. The *shift_oep.hs* file contains the Shift code for the controllers and the simulation setup.

The controller is hierarchically structured. There is a team controller and a 'vehicle controller' for each UAV in the team. The controllers are hybrid automata. The discrete or logical state of the team controller maintains (among other things) the identity of the set of UAVs in the team, the status (functioning or not), role (reserve or attack) of each UAV in the team; the satisfaction or not of the dependency conditions. The discrete state of the vehicle controller maintains the status of component controllers (sensors, weapons)and dependency conditions; its continuous state mirrors the position, speed etc. of the vehicle, obtained from the OEP.

A major advantage of the Shift design is that the *same* team and UAV controllers work for teams with different number of UAVs; moreover, the team controllers are instantiated 'on the fly' in case the team is reconfigured (see section 2.7). It is not possible to design such controllers in Simulink or Matlab.

In *Step 3*, the user presses the *execute program* button **!** to compile the input file *paths.hs* together with the file *shift_oep.hs* and to execute the Shift program. After initialization, the Shift program creates a control window that enables the user to control the advance of simulation time, to inspect the state of all components, and to define breakpoints. Figure 9 displays this control window and the OEP map and figure 10 displays the state of one sub-task controller.

In *Step 4*, the user starts the simulation by pressing the *start button* in the Shift control window. From that point onwards the user controls the advance of simulation time, monitors the state of all of the components in the simulation as well as the estimate of state of the world provided by the database (module 4), which runs as a separate program, and intervenes whenever he is prompted to do so by an exception window generated by the controllers or when new information is received from the database. In both cases, it may happen that replanning is needed, as indicated by the 'feedback' loop in figure 2. In this case, the user terminates the current simulation and goes through the planning-execution cycle again.

[3]Shift is a modeling language and execution system to describe networks of hybrid automata [1]. Shift allows hybrid automata to interact through dynamically reconfigurable input/output connections and synchronous composition.

```
type task_simulation
{
  output
    ucav u1, u2, u3, u4;
          leg leg1, leg2, leg3, leg4, leg5, leg6, leg7, leg8;
          subtask subtask1, subtask2;
          task_controller ctarefa1;
          task tarefa1;
          set(ucav) team1:={}, team2:={};

  state
    number t; // time

  flow default {t' =1;};

  discrete
    i0, i1, i2, i3, i4, normal;

  transition
    i0 -> i1 {} do                                                          // create ucavs with a control structure
          {
                    u1 := create(ucav, p:= small_combo_1);
                    u2 := create(ucav, p:= small_combo_2);
                    u3 := create(ucav, p:= small_combo_3);
                    u4 := create(ucav, p:= small_combo_4);
          },
    i1 -> i2 {} do                                                          // create all legs
          {
            leg1:= create(leg, path:= [ [[93517.725, 111320.00],
                                        [150000.00, 158235.2],
                                        [151000.00, 158000.00]],
                                        [[93517.725, 111320.00],
                                        [150000.00, 158235.2],
                                        [151000.00, 158000.00]]],
                                        vehicles:=[u1,u2], p:=medium_sam12);
            leg2:= create(leg, p_attack:= [], p:=long_sam5_trk);
            leg3:= create(leg, p_attack:= [ [230679.38, 135172.16],
                                            [ 231679.38, 135172.16 ]], p:=medium_sam13);
            leg4:= create(leg, p_attack:= [ [246924.42 , 134016.28],
                                            [ 246924.42, 151410.03]], p:=medium_sam15);
            leg5:= create(leg, p_attack:= [], p:=long_sam6_trk);
            leg6:= create(leg, path:= [ [[93517.725, 111320.00],
                                        [108000.00 , 240000.00]],
                                        [[93517.725, 111320.00],
                                        [108000.00 , 240000.00]]],
                                        vehicles:=[u3,u4], p:=long_sam8_trk);
            leg7:= create(leg, p_attack:= [], p:=long_sam7_trk);
            leg8:= create(leg, p_attack:= [], p:= medium_sam14);
            team1:={u1, u2};                                                // create teams to execute subtasks
            team2:={u3, u4};
          },
    i2 -> i3 {} when (t>1) do                                               // creates subtasks and leg dependencies
          {
            subtask1:= create(subtask, p:=[leg1, leg2, leg3, leg4, leg5], team:= team1);
            requires(leg3):={leg6};
            subtask2:= create(subtask, p:=[leg6, leg7, leg8], team:= team2);
          },
    i3 -> i4 {} do {tarefa1:= create(task, s:=[subtask1,subtask2]);},       // creates task
    i4 -> normal {} do {ctarefa1:= create(task_controller, t:=tarefa1);};   // creates task controller
}
```

Figure 8: Excerpt of execution input file, sub-tasks, legs, leg precedences, and teams.

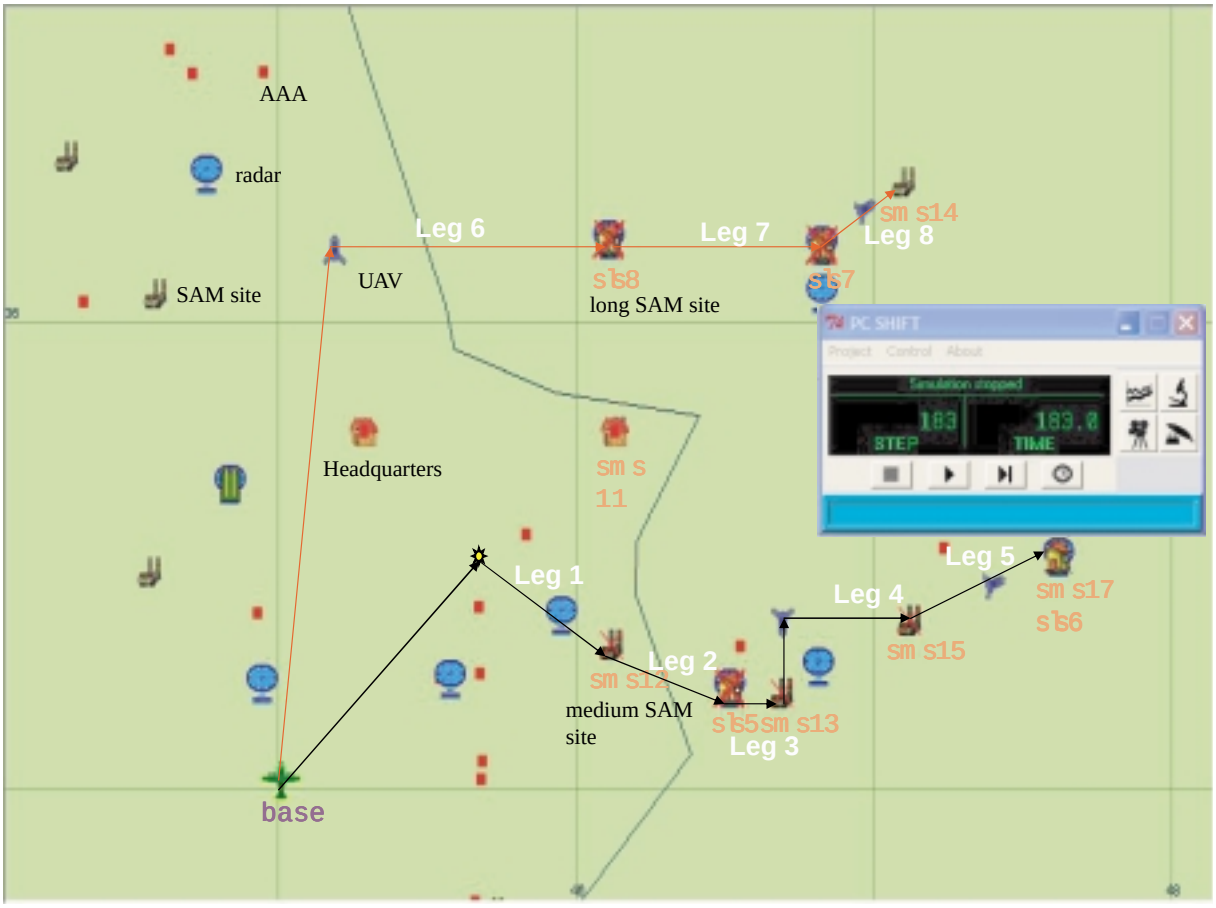


Figure 9: Execution environment.

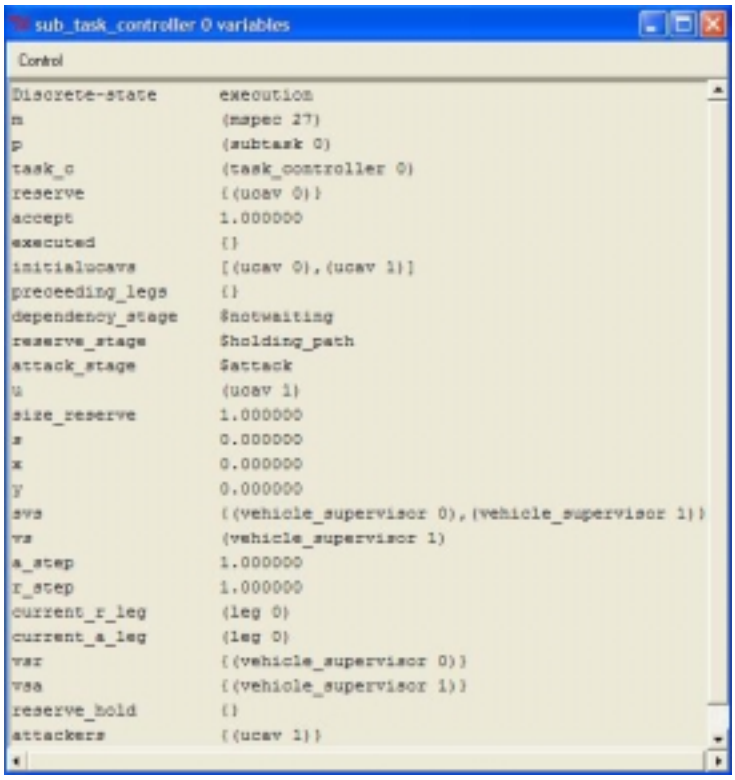


Figure 10: Inspection window.

2.4 Module 4: Database

This module calculates the probability distribution of the Red force conditioned on the observations made by Blue sensors. The following model is used. The Red force is ground based, consisting of SAM launchers, SSM launchers, etc. These launchers have associated radars with various ranges. We do not consider mobile launchers.

A Red force with N targets is then described by a set of the form

$$\textit{Targets} = \{\textit{target}_1 = (\textit{type}_1, (x_1, y_1)), \cdots, \textit{target}_N = (\textit{type}_N, (x_N, y_N))\}.$$

Here the ith target is of $\textit{type}_i$, one of a finite set of *TargetTypes* such as SAM, SSM or radars of different kinds, and (x_i, y_i) is the two-dimensional location of $\textit{target}_i$.

Because the threat is described by a set of targets, knowledge about the threat is represented by a probability distribution of the *set-valued* random variable *Targets*. The prior knowledge of *Targets* is given by the IPB. This knowledge is updated when Blue sensor observations are made. Let the kth sensor observation be (symbolically) denoted by Y_k and let $Y^k = (Y_1, \cdots, Y_k)$ denote the observation history.

The module recursively calculates the conditional probability distribution using Bayes rule:

$$P[\textit{Targets} = \tau \mid Y^k] = \frac{P[\textit{Targets} = \tau \mid Y^{k-1}] P[Y_k \mid \textit{Targets} = \tau]}{\sum_\sigma P[Y_k \mid \textit{Targets} = \sigma] P[\textit{Targets} = \sigma \mid Y^{k-1}]} \tag{1}$$

Above, σ ranges over all possible *Targets*. The recursion is initialized by the prior distribution of *Targets* obtained from the IPB. Equation (1) is derived under the assumption that Y_k and Y^{k-1} are conditionally independent given *Targets*.

Although easy to write down, this recursion is extremely difficult to compute. Suppose there are 3 target types, 100 locations, and up to a total of 10 targets. Then *Targets* is a 3,000-dimensional random vector!

To make the computation tractable, several independence assumptions are made, which reduces the complexity of the probability distribution in terms of both storage requirements and calculating the recursion. The module stores the distribution in a database, which is updated when sensor measurements are made. Calculations of risk functions (needed by the ITP) are then carried out by querying the database.

2.5 Module 5: Java

The java interface to the OEP implements two main functions: serving as the interface between the task execution module (Shift) and the OEP, and making the necessary calls to update the database with new threat distribution values and risk function values. The database layer access is made available through two RMI (Remote Method Invocation) services—Probability Map Generator service and the Risk Function builder service. These services have to be manually started by the user before initiating task execution. When the Shift initialization takes place, it creates an internal mirror of the current state of the OEP by querying the OEP through the java client. The Java Native Interface (JNI) is used for this purpose. During Shift initialization, the following actions are taken:

- An instance of the JVM (Java Virtual Machine) is created and initialized. After the JVM is initialized, one instance of the Java client is created corresponding to each platform in the OEP scenario;
- A static connection to the OEP is established through the naming service;
- Subsequently, connections to the database and the RMI services are established;
- The java client reads the initial state of the OEP, and the threat distribution for the IBP (Initial Battle Plan) is loaded into the database. At this point, the initial value of the risk function is also calculated for the scenario area (the probability map calculator service triggers this calculation in the risk function builder service);
- The java client publishes three methods: get(), set() and a static run() method, which the Shift controllers call every time step (the value of the time step is determined by the argument to the SimulationInterface.runFor() method call). In the get method, the current state of the OEP platform is read into Shift, and the OEP mirror inside the Shift runtime is updated accordingly. The controllers within Shift determine the next step to be taken for that particular platform. These values are then passed to the OEP by calling the set() method in the java client. Once the set() method has been called for all instances of the java client in memory, the run() method is called and execution proceeds in the OEP for one time step.

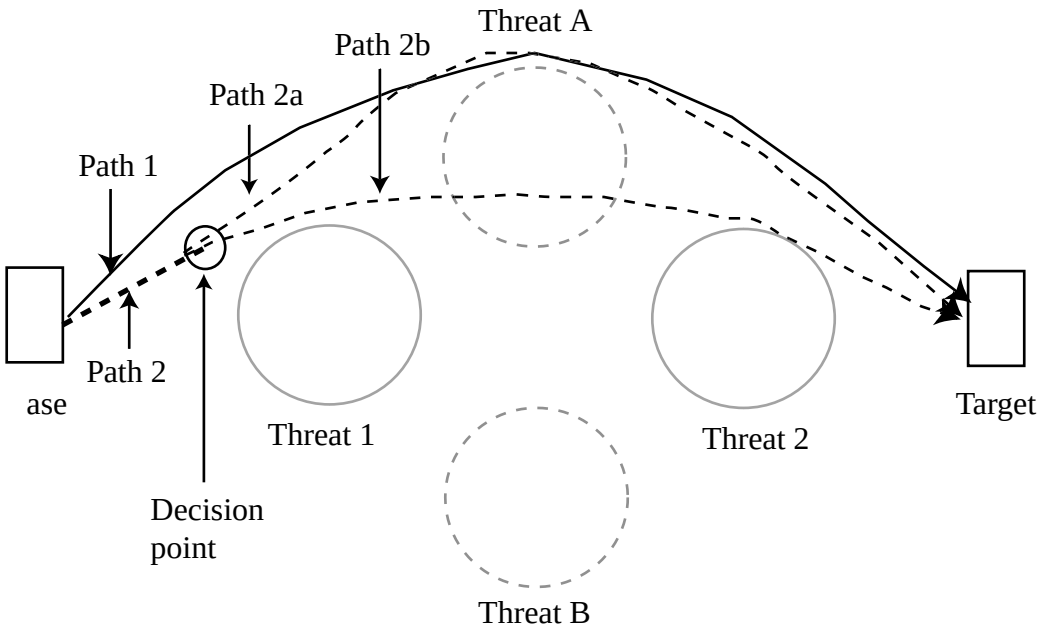


Figure 11: The ITP produces the deterministic path, path 1. A robust path planner produces a policy: follow path 2a if threat A is present, and follow path 2b if threat A is absent.

2.6 Module 6: Robust path planning

Figure 11 displays a scenario with one target, four threats, Threats 1, 2, A and B. The circles denote the threat range. Suppose the presence of Threats 1 and 2 is known with certainty, but there is a chance that Threats A and B are not present. Based on the prior information in the IPB, however, the ITP generates path 1 as the minimum risk path from the base to the target. This path avoids all the threats.

Suppose that when a UAV comes close to the locations of Threats A and B, sensor information reveals whether those threats are actually present, i.e. the prior uncertainty is eliminated. In that case there is a better 'feedback policy' than following the deterministic path 1. This policy would follow the dashed path up to the location 'decision point'. At that point, the UAV would follow path 2a if the sensor information reveals the presence of Threat A, and path 2b if Threat A is absent.

If the resulting flight is along path 2a, it would be worse (more time and fuel consuming) that path 1; if it is along path 2b, it would be better than path 1. However, the contingent path 2 (path 2a or 2b) would on average be better than path 1. The 'robust path planning' module determines paths according to such feedback policies. The module adopts the following model.

Space is discretized with locations denoted x. Time is discretized with index $t = 1, \cdots, T$, in which T is the time horizon of interest. There is a set of m contingent threats, modeled as a Markov chain with states $(s = (s_1, \cdots, s_m)) \in S$, and $s_i \in \{0, 1\}$, indicating absence ($s_i = 0$) or presence of the ith threat; the transition probabilities are $P[s_i(t+1) = 0 \mid s_i(t) = 0] = p$ and $P[s_i(t+1) = 1 \mid s_i(t) = 1] = q$. The threat processes

$s_1, \cdots, s_m$ are all independent, so

$$P[(s(t+1) \mid s(t)] = \prod_i P[s_i(t+1) \mid s_i(t)].$$

The information structure is that a UAV at location x at time t knows the state $s_i(t+1)$ of all threats i located within its sensor range R of x. It is assumed that in one time step a UAV does not travel beyond distance R, so that it always knows whether it will encounter a threat in the next step.

The last element of the model is a 'one-step' cost function $c(s, x, y)$ which is of the form

$$c(s_x, x, y) = \begin{cases} \infty, \text{ path from } x \text{ to } y \text{ goes through the range of threat } s_x \\ |x - y|, \text{ otherwise} \end{cases} \tag{2}$$

In (2), x is the current location, and y ranges over locations that can be reached from x in one time step. Hence the only relevant components of s are those within the sensor range of x, denoted by s_x.

Suppose the UAV at time t is at x, and the threat state is s. Consider the dynamic programming recursion

$$V(s_x, x, t) = \min_y \{c(s_x, x, y) + \sum_{s'} V(s'_y, y, t+1) P[s' \mid s]\}. \tag{3}$$

The minimization in (3) is over all locations y that can be reached in one step from x.

The module solves this recursive equation 'backwards' using the boundary condition $V(s, z, t) \equiv 0$, at the target location z. Having solved this it obtains the next location y at time $t+1$. At time $t+1$ the state s_y becomes known, and a new recursion is created. Evidently, the path by this policy will depend on what information is obtained. Figure 11 shows two possible path realizations, path 2a and path 2b.

To implement the recursion (3), the transition probabilities P must be known, which may not be the case. The 'robust' path planner instead assumes some bounds on these probabilities, represented by $P \in \mathcal{P}$. The robust recursion replacing (3) is

$$V(s_x, x, t) = \max_{P \in \mathcal{P}} \min_y \{c(s_x, x, y) + \sum_{s'} V(s'_y, y, t+1) P[s' \mid s]\}. \tag{4}$$

2.7 Module 7: Dynamic team formation

Figure 12 illustrates a simple example of an ITP output. The set of targets is divided into waves, wave 1, ... , wave N. The 'configuration and schedule' module assigns the targets labeled 1 in the N waves to sub-task 1, those labeled 2 to sub-task 2, and so on. The precedence relation is that all targets in Wave i must be destroyed before any target in Wave $i+1$ is attacked. The final targets (in wave N) are the primary targets.

Four teams are created, one for each sub-task, and team i is allocated to sub-task i, $i = 1, \cdots, 4$. We refer to this as a *static* allocation, because the composition of the team is determined before task execution.

By contrast, a *dynamic* allocation is one in which, after a team attacks a target, the UAVs in the team may be reallocated to other teams. The static allocation is indicated by the solid arrows in the figure; the dynamic allocation is indicated by the dashed arrows. We explain why a dynamic allocation is superior when there is significant uncertainty in the outcome of a sub-task execution.

When a team attacks a target, some resources of the team will be consumed; the resources may be weapons, fuel, or a UAV. The resource consumption is random. Suppose the objective is for each team to reach its primary or final target with a specified level of resources. For example, team 1 must reach its final target with at least 2 UAVs with a certain number of weapons, with probability at least 0.5. Given the resource consumption rates, we can then work backwards to figure out the resources needed by team 1 at the beginning of wave $N-1$ so that it can meet the resource requirements for the primary target. Proceeding in this way, we arrive at the initial composition of team 1.

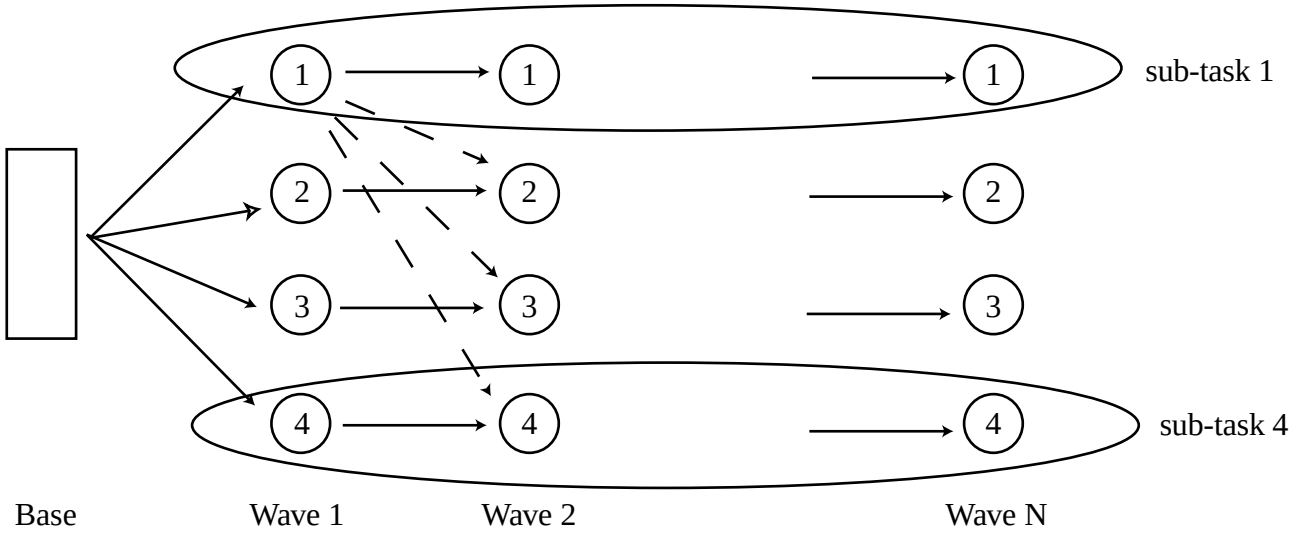


Figure 12: The ITP output groups targets into waves. A static allocation assigns a fixed team i to sub-task i, $i = 1, \cdots, 4$. A flexible allocation re-allocates teams after each target strike.

Consider a simple example with $N = 4$, and suppose that in attacking a target there is a 0.25 probability that the UAV is destroyed. We require that at least 2 UAVs reach the final target, with probability at least 0.5. Observe that a UAV's probability of survival after attacking 3 targets is $(0.75)^3 \approx 0.4$. An elementary calculation then shows that in a static allocation, a team must begin with 6 UAVs to ensure that at least 2 UAVs reach the final target (with probability at least 0.5). So for the four sub-tasks, a total of 24 UAVs is needed under a static allocation to achieve the objective.

More generally, if p_0 is the probability of destruction of a UAV in each stage, the probability that a UAV reaches the final target is $p = (1-p_0)^N$. If n UAVs are launched, the probability that at least k of them reach their final targets is

$$sum_{m=k}^{n} \binom{n}{m} p^m (1-p)^{n-m}.$$

On the other hand, if at the end of each target strike, we can reallocate UAVs from teams that suffered a lower attrition to teams that had heavier losses, only 12 UAVs would be needed—a resource savings of 50 percent. Evidently, the more uncertain is the resource consumption, the greater are the potential savings from dynamic (re)allocation of teams.

However, reallocation of UAVs may itself be costly. For example, if the teams are located far from each other, time and fuel will be consumed by rellocation. Perhaps a greater cost may be the difficulty faced by the people who ultimately control a team in dealing with dynamically reconfigured teams. Thus in dynamic allocation, a balance must be reached between the savings from reallocation and its cost. The 'dynamic team allocation' module proposes a mathematical model of this balance, together with an algorithm that determines, at the end of each target strike, the team re-allocation.

The module takes as input: (1) the wave structure similar to that shown in figure 12; (2) the survival probability; and (3) the cost of reallocating a UAV from one team to another. It produces as output an allocation 'policy', which reallocates surviving UAVs at the end of each target strike.

2.8 Module 8: Path planning with two constraints

The ITP relies on a fast 'path planner', which determines for a given set of 'origins' $O \subset R^2$ and each 'destination' $x \in R^2$ a path γ that

$$\min_{\gamma \in \Gamma} \rho(\gamma) = \int_{\sigma=0}^{\tau} f(\gamma(\sigma)) d\sigma. \tag{5}$$

Here Γ is the set of all paths that start at some origin $\gamma(0) \in O$[4] and end at the destination $\gamma(\tau) = x$; $f(z)$ is the 'risk' per unit distance incurred at location z; and σ parametrizes the path length. The minimum risk $V(x)$ incurred in reaching x satisfies the *eikonal* equation [3]

$$|\nabla V(x)| = f(x), \quad V(x) = 0, x \in O.$$

The path planner calculates the 'value' function V, whose contour plots are displayed in the ITP window of figure 5. Also, given any destination x, the path planner calculates the minimum risk path from O to x; the ITP window displays some of those paths.

It may be necessary or useful to find the minimum risk path subject to another path constraint. For example, one may want to constrain the minimum risk path search to those paths that can be completed with say a specified amount of fuel. Such a constrained optimal path is a solution of the problem

$$\min_{\gamma \in \Gamma} \rho(\gamma) = \int_{\sigma=0}^{\tau} f(\gamma(\sigma)) d\sigma \tag{6}$$

$$\text{subject to } c(\gamma) = \int_{\sigma=0}^{\tau} g(\gamma(\sigma)) d\sigma \leq C. \tag{7}$$

In (7) $g(x)$ is the rate of fuel consumption and C is the constraint on total consumption.

[4]The set of origins at the end of each wave consists of the blue base and the locations of targets at preceding waves.

The solution of (6)-(7) is much more difficult than the unconstrained problem (5). We can convert the constrained problem to a family of unconstrained problems parameterized by a Lagrange multiplier λ,

$$\min_{\gamma\in\Gamma} L_\lambda(\gamma) = \int_{\sigma=0}^{\tau} [f(\gamma(\sigma)) + \lambda g(\gamma(\sigma))]d\sigma. \tag{8}$$

Observe that the larger is $\lambda > 0$ the greater is the weight placed on the g-component of the cost in $L_\lambda(\gamma)$ of path γ.

The iteration works as follows. One selects a weight $\lambda > 0$, obtains the path γ^*_λ that minimizes $L_\lambda(\gamma)$ and checks whether the constraint (7) is satisfied at γ^*_λ. If it is satisfied, λ is reduced; if it is not satisfied, λ is increased. The best λ is the smallest for which the constraint is satisfied. So we have an iterative method for solving a constrained optimization problem using "soft" constraints, i.e. a Lagrange multiipler formulation. One iterates on the Lagrange multiplier.

3 Module 1: Interactive task planner

This section discusses the interactive task planner or ITP, which was briefly described in section 2.1. We recall the setting. The Blue force comprises a set of UAVs, each of which can be configured with different capabilities (sensors, weapons, etc). There is a set of targets called the Red force. The Blue force is used to attack the Red force, which threatens the attacking Blue force. With the help of the ITP the planner organizes Blue's attack into a *plan*. The plan is further specified by the 'configuration and schedule' module, after which it is executed by the Shift control module (module 3). The ITP module may be reinvoked during execution, as indicated in figure 2.

We will formally describe a plan 'design space', and performance measures to compare plans. The formalization permits the development of an algorithm that automatically generates good—even 'optimal'—plans. The ITP software implements this algorithm. Of course, considerations beyond those captured in the performance measures will enter into the specification of the final plan. The planner introduces these considerations by interacting with the ITP software to modify the automatically generated plan.

3.1 Threat

We now introduce some terminology for use in a mathematical description of the threat. *Target* is a generic term for Red force entities of different types such as SAM launchers, SSM launchers, radars, etc. There is a finite set of *types*, called *TargetTypes*. We will not be concerned with mobile targets. So a target is completely characterized by its type and its (two-dimensional) location (x, y). A Red force with N targets is thus fully described by a set of the form

$$\textit{Targets} = \{\textit{target}_1 = (\textit{type}_1, (x_1, y_1)), \cdots, \textit{target}_N = (\textit{type}_N, (x_N, y_N))\}. \tag{9}$$

There may be uncertainty about the Red force. We adopt a Bayesian view, which summarizes prior knowledge about the Red force as a probability distribution of the *set-valued* random variable *Targets*. The prior knowledge of *Targets* is given by the IPB. We denote this initial distribution (at time $t = 0$) by $P_{\textit{threat}}(0)$.[5] Two examples will help illustrate $P_{\textit{threat}}(0)$.

Example 1. The IPB indicates that the Red force consists of one SSM at a known location $(\bar{x}, \bar{y})$, four SAM sites at unknown locations in area A_1, and six SAM sites at unknown locations in area A_2. In this example $N = 11$, so

$$\begin{aligned} P_{\textit{threat}}(0)(\textit{Targets} &= \{(\textit{type}_1, (x_1, y_1)), \cdots, (\textit{type}_{11}, (x_{11}, y_{11}))\}) \\ &= \prod_{i=1}^{11} P_i(\textit{type}_i, (x_i, y_i)), \end{aligned} \tag{10}$$

in which

$$P_1(\textit{type}_1 = (SSM, (x, y))) = \begin{cases} 1, & (x, y) = (\bar{x}, \bar{y}) \\ 0, & \text{otherwise} \end{cases},$$

[5] This probability distribution is updated during execution when Blue sensor observations are made; see sections 2.4, 6.

and the distributions $P_2, \cdots, P_{11}$ have densities

$$p_i(\textit{type}_i = SAM, (x,y)) = \begin{cases} |A_1|^{-1}, & (x,y) \in A_1, i = 2, \cdots, 5 \\ |A_2|^{-1}, & (x,y) \in A_2, i = 6, \cdots, 11 \\ 0, & \text{otherwise} \end{cases} . \tag{11}$$

In (11), $|A_i|$ is the area of A_i. The 'product' form (10) is obtained under the assumption that the 11 targets are independent random variables. To obtain (11) it is assumed that the location of target i is uniformly distributed over area A_i.

Example 2. There is more uncertainty than in Example 1: The number N_1 of SAMs in A_1 and the number N_2 of SAMs in A_2 are independent random variables. So

$$\begin{aligned} P_{\textit{threat}}(0)(\textit{Targets}) = P_1(\textit{target}_1) \quad & \times \quad \prod_{i=1}^{N_1} p_i(\textit{type}_i, (x_i, y_i)) P_1(N_1) \\ & \times \quad \prod_{i=1}^{N_2} p_i(\textit{type}_i, (x_i, y_i)) P_2(N_2); \end{aligned}$$

$P_1(\textit{target}_1)$ is the same as before, p_i is the uniform density over A_i (given by (11)), and $P_i(N_i)$ is the probability distribution of N_i, $i = 1, 2$.

In principle, the distribution $P_{\textit{threat}}(0)$ can be quite complicated. We will suppose, however, that the IPB takes on the following more restrictive form:

The Red force is distributed over areas $A_1, \cdots, A_k$. In area A_j there are N_{tj} targets of type $t \in$ *TargetTypes* whose locations are independently and uniformly distributed. The random number of targets N_{tj} are all independent with distribution $P_{tj}(N)$.

This restriction implies that the IPB is represented as a distribution of the form

$$P_{\textit{threat}}(0)(\textit{Targets}) = \prod_t \prod_{j=1}^{k} \prod_{i=1}^{N_{tj}} p_{tj}(\textit{type} = t, (x_i, y_i)) P_{tj}(N_{tj}), \tag{12}$$

in which t ranges over *TargetTypes*, and

$$p_{tj}(\textit{type} = t, (x_i, y_i)) = \begin{cases} |A_j|^{-1}, & (x_i, y_i) \in A_j \\ 0, & \text{otherwise} \end{cases} . \tag{13}$$

Examples 1, 2 above are special cases of this model. The major restriction implied by model (12)-(13) is the statistical independence of the threats in the different areas. This restriction seems reasonable.[6]

[6]The restriction can of course be removed and one may admit any distribution of the random variable *Targets*.

The probability distribution (12)-(13) is completely specified by the list of areas $A = \{A_1, \cdots, A_k\}$ and the random vector $N = \{N_{tj}; t \in \textit{TargetTypes}, 1 \leq j \leq k\}$. Hence we will sometimes use the shorthand notation $P_{A,N}$ to denote this distribution.

A target poses a *threat*, over a specific region, which is a circle centered at the target location and radius that depends on the target type. A UAV in this threat region will be destroyed with a certain probability that depends on the target type and the heading of the UAV (which determines the signature that it exposes to the target), as well as the amount of time that the UAV is in the region. We will assume that a numerical value can be assigned to the threat posed by a target at any location. Section 3.3 gives examples of such numerical values and shows how they are used to calculate the *risk* incurred by a UAV as it traverses a path that goes through threat regions.

For the OEP, figure 4 lists some targets, their OEPid, OEPtype, location, and range. For example, target ew1 is an early warning radar of type ew_radar_site_type, located at (83.1, 357.8); it poses no threat (range is 0), because it contains no weapons. On the other hand, target long_sam1 has a threat range of 80 km.

3.2 Plan design space

With the help of the ITP, the planner produces a plan, which organizes Blue's attack into a set of *sub-tasks*, each of which is a list of targets to be destroyed. The plan must determine the set of objectives, hence the set of sub-tasks. We assume that the planner is given a set of *primary* targets, and so these must be included in the plan. The plan will typically include additional targets.

A UAV dispatched to attack a target will fly over a *path* γ, during which it will incur a certain *risk* $\rho(\gamma)$, which can be translated into the probability that the UAV will be destroyed along its flight path. We assume that the planner selects (or is given) a maximum risk threshold ρ_{max}.[7]

An acceptable or feasible plan is a set of targets, $\textit{TargetList} = \{\textit{target}_1, \cdots, \textit{target}_n\}$, together with a set of paths $\textit{PathList} = \{\gamma_1, \cdots, \gamma_n\}$, one for each target, such that the risk along all these paths $\rho(\gamma_i) \leq \rho_{max}$. We tenatively define the plan design space as the set of all feasible plans. However, this is incomplete for three reasons.

First, we require all primary targets to be included in *TargetList*. Second, the plan must group the targets into a set *SubtaskList* of *sub-tasks*, each of which will be assigned to a team of UAVs. The third reason is more subtle. The risk $\rho(\gamma)$ of a path depends on which targets have already been destroyed. *Thus the risk $\rho(\gamma)$ depends on the order in which targets are destroyed.* So the plan must also specify this order or *precedence relation*, denoted by $\succ$, with the intepretation that

$$\textit{target}_i \succ \textit{target}_j$$

means that $\textit{target}_i$ must be attacked (and destroyed) before $\textit{target}_j$.

A *feasible plan* is a 4-tuple $\textit{plan} = (\textit{TargetList}, \textit{SubtaskList}, \textit{PathList}, \succ)$, in which

(1) $\textit{TargetList} = \{\textit{target}_1, \cdots, \textit{target}_n\}$ is a set of targets, and $\textit{PathList} = \{\gamma_1, \cdots, \gamma_n\}$ are their paths;
(2) *SubtaskList* is a partition of *TargetList* into of tasks, and

[7] In figure 5, this is the entry in the window called 'Wave Threshold'.

(3) $\succ$ is a precedence relation or partial order on *TargetList*, so that for all $i = 1, \cdots, n$

$$\rho(\gamma_i) \leq \rho_{max}. \tag{14}$$

In (14), the risk $\rho(\gamma_i)$ is calculated under the assumption that all targets $target_j \succ target_i$ have been destroyed.

The risk R associated with a plan is the maximum risk incurred along any path,

$$R(plan) = \max_i \rho(\gamma_i). \tag{15}$$

The *plan design space* is the space of all feasible plans. An optimal plan has minimum risk,

$$R(plan^*) = \min\{R(plan) \mid plan \in plan\ design\ space\}.$$

The ITP implements a procedure that can find an optimal plan. Before we describe this procedure, we develop a model to quantify the risk along a path.

3.3 Risk along a path

The risk incurred by a UAV flying along a path $\gamma(\sigma), 0 \leq \sigma \leq \tau$, from a specific origin $\gamma(0) = o$ to a destination $\gamma(\tau) = d$, depends on the threat distribution P_{threat}. We describe how we calculate this risk, $\rho(\gamma)$. We begin with an example.

Example 3. P_{threat} indicates SAM sites at known locations $(x_i, y_i), i = 1, \cdots, n$. A UAV flies from its base at location $(0, 0)$ to $(\bar{x}, \bar{y})$ at a fixed speed v (normalized to $v = 1$) along the path $\gamma(\sigma), \sigma \in [0, \tau]$, as indicated in figure 13. At any point along its path, the threat posed by a SAM site depends on how far the UAV is from the site. We suppose that the resulting risk along γ is quantified as

$$\rho(\gamma) = \sum_{i=1}^{n} \int_{\sigma=0}^{\tau} f(|\gamma(\sigma) - (x_i, y_i)|) \frac{d\sigma}{v}. \tag{16}$$

In (16), $|\gamma(\sigma) - (x_i, y_i)|$ is the (Euclidean) distance between $\gamma(\sigma)$ and (x_i, y_i); $f(d) \geq 0$ is any decreasing function of d that measures the *'instantaneous' risk*; and σ parametrizes the path.[8]

One choice for f is

$$f(d) = \begin{cases} 1, & d \leq D \\ 0, & d > D \end{cases} \tag{17}$$

where D is the threat range of a SAM. A more complex choice is

$$f(d) = (1 + d)^{-2}, \text{ or } f(d) = e^{-\alpha d}, \text{ for some } \alpha > 0. \tag{18}$$

[8] For the 'fast marching' algorithm described later, f must be a smooth function.

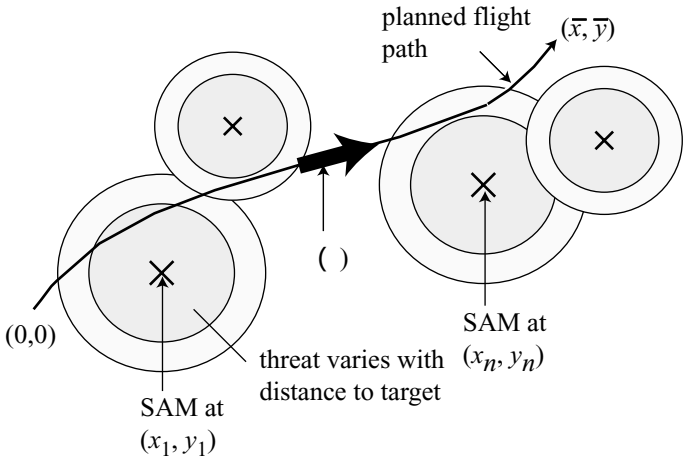


Figure 13: A UAV travels along the path γ encountering threats from the SAM sites at $(x_i, y_i), i = 1, \cdots, n$. The threat depends on the distance of the path from the SAM site.

Evidently, the risk $\rho(\gamma)$ depends on the planned path γ.[9] So the minimum risk is $V(\bar{x}, \bar{y}) = \min_\gamma \rho(\gamma)$, where the minimization is over all paths γ from $(0, 0)$ to $(\bar{x}, \bar{y})$. More formally, for this example, the *value function* V is

$$V(\bar{x}, \bar{y}) = \min_{\gamma} \rho(\gamma). \tag{19}$$

We will see below how to calculate the value function and the risk minimizing path in (19).

We now describe how to calculate the risk along a path for a general threat of the form $P_{A,N}$, with $A = \{A_1, \cdots, A_k\}$ and $N = \{N_{tj}; t \in \textit{TargetTypes}, 1 \le j \le k\}$. (The detailed distribution is given by (12)-(13).)

In the first step, we define the instantaneous risk function at any point (x, y) as

$$r(x, y; P_{A,N}) = \sum_{j=1}^{k} \sum_{t} \sum_{N_{tj}=0}^{\infty} \sum_{n=1}^{N_{tj}} \left[\int_{A_j} f_t(|(x, y) - (x_n, y_n))|A_j|^{-1} dx_n dy_n \right] P(N_{tj}). \tag{20}$$

In (20) $f_t(d)$ is the instantaneous risk posed by a target of type t to a UAV at a distance d from the target. f_t may be of the form (17), (18) or some other form. So the integral in (20) is the expected value of this instantaneous risk posed by a target of type t located at a random point (x_n, y_n) that is uniformly distributed over area A_j. The sum over n is the total risk posed by N_{tj} such targets. The sum over N_{tj} accounts for the random distribution of N_{tj}. The sum over t accounts for different types of targets. Finally the sum over j accounts for all the areas. The argument $P_{A,N}$ in r emphasizes the role of the threat. The next example extends example 3 to a general threat.

[9] If we interpret $f(d)\Delta$ as the probability that the UAV is destroyed in time Δ when it is at a distance d from the SAM, $e^{-\rho(\gamma)}$ is the probability of survival along the path γ.

Example 4. The risk faced by a UAV flying at speed v along a path γ from $(0,0)$ to $(\bar{x},\bar{y})$, facing threat $P_{A,N}$ is

$$\rho(\gamma; P_{A,N}) = \int_{\sigma=0}^{\tau} r(\gamma(\sigma); P_{A,N}) \frac{d\sigma}{v}, \tag{21}$$

in which r is given by (20). Hence the value function for threat $P_{A,N}$ is

$$V((\bar{x},\bar{y}); P_{A,N}) = \min_{\gamma} \rho(\gamma; P_{A,N}). \tag{22}$$

3.4 Value function calculation

The value functions (19) and (22) are special cases of the following problem:

We are given a set O of possible origins, a target destination (x,y), and a nonnegative, instantaneous risk or cost function f. Find

$$V(x,y) = \min_{\gamma\in\Gamma(x,y)} \rho(\gamma) = \min_{\gamma\in\Gamma(x,y)} \int_0^{\tau} c(\gamma(\sigma))d\sigma. \tag{23}$$

Here $\Gamma(x,y)$ is the set of all paths γ which begin at some permitted origin, $\gamma(0) \in O$, and terminate at (x,y), and $\rho(\gamma)$ is the risk along γ. Moreover, we must find the optimal path for any destination.

The value function satisfies the *eikonal* equation

$$|\nabla V(x,y)| = c(x,y), \tag{24}$$

with boundary condition

$$V(x,y) = 0, \quad (x,y) \in O. \tag{25}$$

In (24),

$$|\nabla V(x,y)| = [(\frac{\partial V}{\partial x})^2 + (\frac{\partial V}{\partial y})^2]^{1/2},$$

Moreover, the optimal paths γ^* can be obtained by following the negative of the gradient of V:

$$\frac{d\gamma^*}{d\sigma}(\sigma) = -\nabla V(\gamma(\sigma)) \tag{26}$$

The ITP repeatedly uses a very fast algorithm (called the 'fast marching' algorithm [2, 3]) to solve the eikonal equation (24)-(25). Then, given any destination, it calculates the optimal path by 'steepest descent' (26).

Figure 5 displays the contour plots of the value function and several optimal paths for the threats listed in the scenario window. Observe, as is to be expected, that the optimal paths are orthogonal to the contours.

Remark The cost function c in (23) can be modified to include restrictions on the admissible path. For example, there may be 'no-fly' zones over which a UAV may not fly for strategic considerations or because of geographic obstacles. These restrictions can be easily incorporated in the formulation by setting $c(x,y)$ to a very large value for (x,y) in these no-fly zones. In this way, 'soft constraints' can be included by appropriately modifying the function c.

3.5 Risk of a plan with prespecified order of attack

Suppose the planner has selected a *TargetList* and a particular order of attack,

$$\textit{target}_1 \succ \cdots \succ \textit{target}_n.$$

We can calculate the optimal paths as follows, assuming that the path to the first target must originate at the blue base. Let (x_i, y_i) be the location of $\textit{target}_i$.

Define, originating sets $O_1, \cdots, O_n$,

$$O_1 = \textit{BlueBase}, \quad O_{i+1} = O_i \cup \{(x_i, y_i)\}, \;\; i \geq 1.$$

Let

$$V(x_i, y_i; P_i) = \min_{\gamma \in \Gamma_i(x_i, y_i)} \rho(\gamma; P_i),$$

and let γ_i^* be the optimal path. Here P_i is the threat after $\textit{target}_1, \cdots, \textit{target}_{i-1}$ have been destroyed; and $\Gamma_i(x_i, y_i)$ consists of all paths originating in O_i and ending at (x_i, y_i).

The optimal path γ_i^* to $\textit{target}_i$ starts from the blue base or at a location of a previously destroyed target. The risk, R, incurred by this plan is the maximum or worst-case risk for any sub-task,

$$R = \max\{V(x_1, y_1; P_1), \cdots, V(x_n, y_n; P_n)\}.$$

3.6 The ITP procedure and optimal plan

We now describe the ITP procedure. Suppose that the IPB consists of a set of targets, $\textit{Targets}_1$, with known locations and types. Also given is a subset *PrimaryTargets* that must be destroyed. Lastly, the locations of the Blue base, $BlueBase$, is known; this may be one or more locations from which UAVs may be dispatched.

Step 1 Let P_1 be the threat corresponding to the targets $\textit{Targets}_1$. The planner selects a Wave Threshold of acceptable plan risk, R. The fast marching algorithm computes the value function

$$V_1(x, y) = V((x, y); P_1) = \min_{\gamma \in \Gamma_1(x,y)} \rho(\gamma; P_1),$$

in which $\Gamma_1(x, y)$ is the set of all paths starting at origins $O_1 = \textit{BlueBase}$ and ending at (x, y); $\rho(\gamma; P_1)$ is the risk along γ under the threat P_1, given by (23).

The value function V_1 is evaluated at the locations of all targets in $\textit{Targets}_1$. Let

$$\textit{Wave}_1 = \{\textit{target} \in \textit{Targets}_1 \mid V_1((x, y)_{target}) \leq R\}$$

be the subset of targets at whose locations $(x, y)_{\textit{target}}$ the value function is less than R. This means that targets in $\textit{Wave}_1$ can be detroyed with risk less than R. Let $\textit{PathList}_1$ be the optimal paths to $\textit{Wave}_1$ targets obtained by steepest descent of V_1.

Step 2 Let $\mathit{Targets}_2 = \mathit{Targets}_1 \setminus \mathit{Wave}_1$ be the targets remaining after those in Wave_1 are destroyed, and let P_2 be the threat corresponding to $\mathit{Targets}_2$. Let O_2 be the origins consisting of the *BlueBase*, together with the locations of Wave_1 targets. The fast marching algorithm computes the value function

$$V_2(x,y) = V((x,y);P_s) = \min_{\gamma\in\Gamma_2(x,y)} \rho(\gamma;P_2),$$

in which $\Gamma_2(x,y)$ is the set of all paths starting at origins O_2 and ending at (x,y).

The value function V_2 is evaluated at the locations of all targets in $\mathit{Targets}_2$. Let

$$\mathit{Wave}_2 = \{\mathit{target} \in \mathit{Targets}_2 \mid V_2((x,y)_{target}) \le R\}.$$

After Wave_1 targets have been destroyed, the targets in Wave_2 can be destroyed with risk less than R. Let $\mathit{PathList}_2$ be the optimal paths to Wave_2 targets obtained by steepest descent of V_2.

Step k We continue in this way. In the kth step, we set

$$\begin{aligned}
\mathit{Targets}_k &= \mathit{Targets}_{k-1} \setminus \mathit{Wave}_{k-1},\\
O_k &= O_{k-1} \cup \{\text{Locations of } \mathit{Wave}_{k-1} \text{ targets}\},\\
V_k(x,y) &= \min_{\gamma\in\Gamma_k(x,y)} \rho(\gamma;P_k),\\
\mathit{Wave}_k &= \{\mathit{target} \in \mathit{Targets}_{k-1} \mid V_k((x,y)_{target}) \le R\},\\
\mathit{PathList}_k &= \{\text{optimal paths to } \mathit{Wave}_k \text{ targets}\},
\end{aligned}$$

in which P_k is the threat posed by $\mathit{Targets}_k$, and $\Gamma_k(x,y)$ comprises all paths that start in some location in O_k and end at (x,y).

Stopping condition The process stops at the smallest k for which one of two conditions holds:

$$\mathit{Wave}_{k-1} \neq \emptyset \quad\wedge\quad \mathit{Wave}_k = \emptyset \tag{27}$$

$$\mathit{PrimaryTargets} \quad\subset\quad \cup_{i=1}^{k} \mathit{Wave}_i. \tag{28}$$

Theorem If condition (27) holds, there is no plan that can destroy all primary targets with risk at most R. If condition (28) holds, the plan with

$$\mathit{TargetList} = \cup_{i=1}^{k} \mathit{Wave}_i, \tag{29}$$

$$\mathit{PathList} = \cup_{i=1}^{k} \mathit{PathList}_i, \tag{30}$$

$$\succ \;:=\; \mathit{Wave}_1 \succ \cdots \succ \mathit{Wave}_k, \tag{31}$$

destroys all primary targets with risk at most R. Moreover, this plan is optimal if R is the smallest risk for which (28) holds.

The top diagram in figure 14 gives a schematic representation of the results of the plan produced by this procedure. There are four primary targets, indicated by the rightmost, shaded circles. The *TargetList* contains 12 targets, organized in three waves. The arrows denote the origin and destination of the minimum risk paths. The targets in the top row are not in the sub-tasks because they were not reached in the first three waves.

The final element of the plan is the decomposition of *TargetList* into sub-tasks. This can be done automatically or visually. Observe that the targets and paths form a directed tree. The idea is to construct sub-tasks that form chains. In the figure seven tasks are formed.

The ITP procedure provides one additional output, called the *sensitivity tables*, which is used in the refinement of the procedure, described in section 3.7. A sensitivity table entry is defined for each pair of targets $target_i \in Wave_i$ and $target_{i+1} \in Wave_{i+1}$ as the increased risk in the path to $target_{i+1}$ if $target_i$ is *not* destroyed. This is also the reduction in risk resulting from the destruction of $target_{i+1}$.

3.7 Refinement of the ITP procedure: mixed initiative

The procedure just described is fully automated. The planner can intervene in several ways to improve the plan, bringing into consideration other factors that are not part of the procedure.

First, at the end of each step k, the ITP window of figure 5 displays the values $V_k((x,y)_{target})$ for each target. The planner can study these values together with the target locations, and decide to *add* targets to $Wave_k$. These additions of course will incur a risk larger than R, so they must represent to the planner 'targets of opportunity' whose destruction is worth the extra risk.

Second, the planner may *delete* a target from $Wave_k$, perhaps because it does not significantly reduce the risk to targets in $Wave_{k+1}$. The risk reduction is known from the sensitivity table. The reason for deletion might be to conserve blue forces, or to reduce the *SubtaskList* so that the plan can be executed in less time. These two interventions change the *SubtaskList* (29).

Third, the planner may weaken the precendence relation (31), which requires every $Wave_i$ target to be destroyed before any $Wave_{i+1}$ target is attacked. This will reduce the level of coordination needed in sub-task execution at the cost of greater risk, quantified by the sensitivity table.

Fourth, the planner may change the grouping of sub-tasks into tasks, based on considerations of team composition. For example, targets of the same type may be grouped together because they are attacked by similar weapons.

Lastly, the planner may alter one or more paths in *PathList*, which gives the way points for every path.

The bottom diagram in figure 14 shows the plan resulting from modifications made by the planner: Three targets have been deleted from *SubtaskList*; the precedence relation is weakened; and sub-tasks have been regrouped into four tasks.

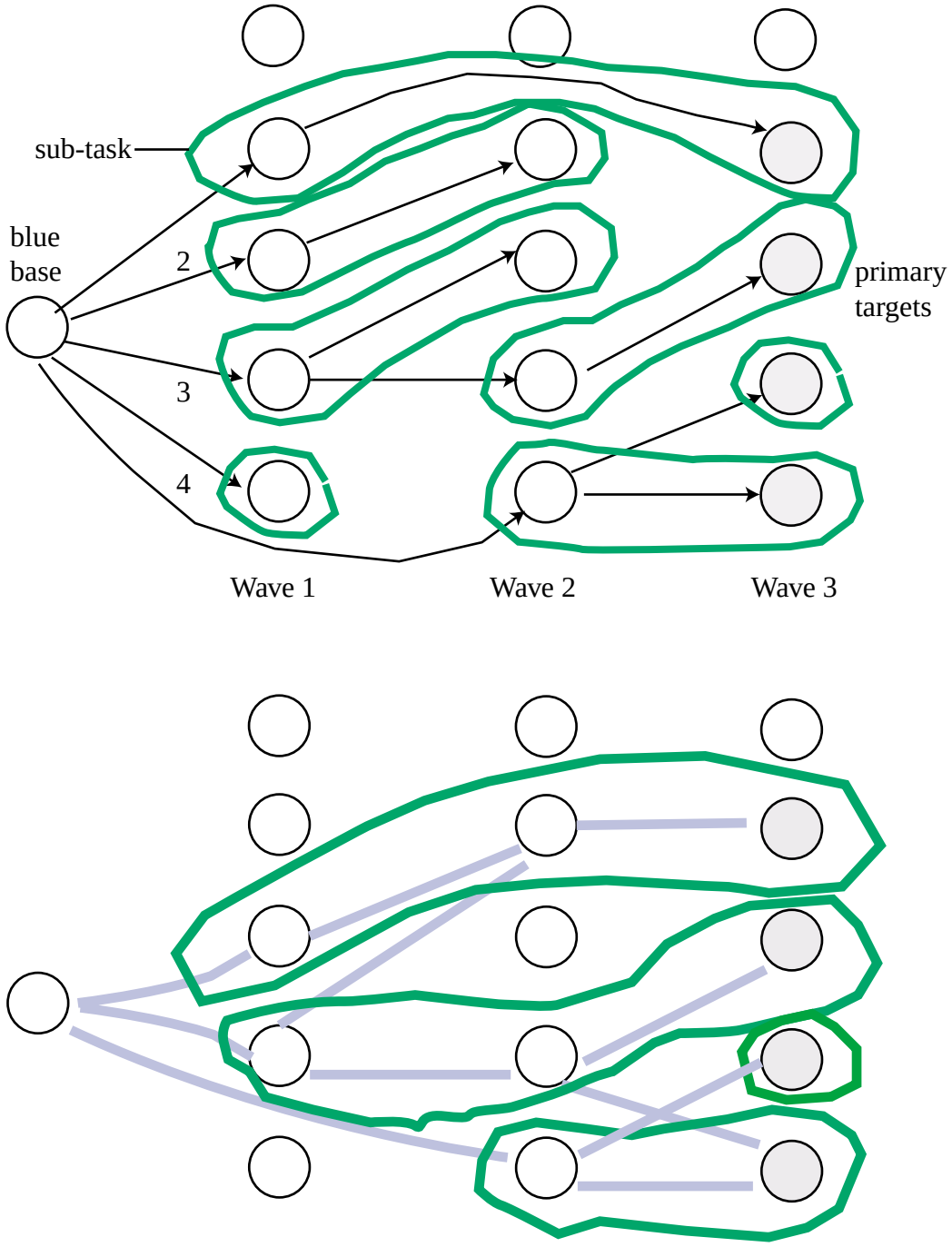


Figure 14: Circles denote all targets. Top diagram: The ITP procedure generates $Wave_1, Wave_2, Wave_3$; the arrows denote origin and destination of optimal paths; the targets are grouped into seven sub-tasks. Bottom diagram: The result of modifications by the planner.

4 Module 2: Configuration and schedule

Semi- or fully automated team composition and tasking (TCT) is central to increasing the ratio of UAV to human handlers (HH). Such automation transfers complex lower level decision functions to the automata (system level) and enables the HH to concentrate on higher level operational decisions (tactical, long-term resource management, etc.). It also makes possible autonomous task execution by enabling automated re planning during task execution. This section outlines the principles and an algorithm that automates the process of TCT.

4.1 Task scheduling in the MICA context

In broad terms, scheduling is the process of selecting, organizing, and timing resource usage to carry out all the activities required to produce some desired outcome at desired times. Tasks have clearly defined Primary Objectives (PO) and additional Secondary Objectives (SO) that are added to enhance the probability for success of a task plan. The tasks are made up of Subtasks, and each Subtask is assigned to a single Team. If a task is described as a grouping of actions leading to a defined outcome, then the task's objectives are the minimum set of purposeful actions whose desired timely outcome renders the task's successfully completed. As such, objectives in the MICA context may be a search maneuver, a bomb run on a target, a refueling maneuver, a cooperative jamming maneuver (providing jamming for other platforms), etc.

Task Categories
The difficulties in applying scheduling consistently and broadly when dealing with a set of tasks is in defining common underlying principles that are sufficiently general to allow repeated application for all tasks in the set. Our first goal then is to identify categories of tasks based on commonalities in planning and execution requirements. For each such category, we can then develop a scheduling model that can be applied for any task within the category.

In the MICA context, tasks may be classified in three categories:

- Preplanned Strike on known targets (main objectives are known before mission start);
- Intelligence Gathering on known enemy assets or regions (an asset or a region is marked for gathering of additional information);
- Loitering strike or intelligence gathering on time critical targets of opportunity (preemptive response in anticipation of future enemy activity or in an attempt to deny the enemy the use of options).

The focus of our research has been on a SEAD task which fits into the ”Preplanned strike” category. It should be stated that similar formulations can be defined for other categories with little additional effort.

Scheduling in this context may involve the following four steps with iterations:

1. Establishing prosecution partial order chains and inter-chain dependencies; efficient allocation of limited resource;
   (a) Appropriate weapons selection for each objective (target);

(b) Appropriate sensors selection for each objective;

(c) Analysis of attrition and the addition of reserves;

(d) Efficient platforms selection to deliver all above resources (sensor or weapon) to assigned locations for use in some scheduled action;

2. Timeline generation - Evaluation of all mission timeline for compliance with an overall timetable.

Premises and Underlying Planning Strategy
We state our premises or underlying strategy for adjustable system autonomy regarding the task planning and plan execution because it shapes our approach and distinguishes our solutions from others'. The following is a summary of some of the distinguishing elements in our approach to task planning and execution:

1. Human handlers always have the responsibility for formulation of Commander's intent in the form of generating tasks with defined primary objectives and scheduling goals (see more on scheduling goals bellow);

2. Human handlers monitor re planning solutions and resolve planning exceptions (planning exceptions are planning solutions that do not meet all task objectives and/or planning goals);

3. The system responds to maintain a plan for prosecuting the primary objectives (at all levels of the control hierarchy). It notifies the human handler when the plan is no longer executable and gives alternative plan that can prosecute remaining objectives with the remaining assets or it gives the next best plan that requires additional assets.

4. Re-planning is incremental:

   (a) At platform level trajectories are recalculated to overcome delays, a target may be reengaged with a second weapon, etc;

   (b) At the team level assignments may be swapped to make up for delays and attrition or the Subtask may be re planned;

   (c) At the task level sub-tasks primary objectives may be reshuffled, task may be re planned, or the Commander's intent (primary objectives and scheduling goals) may be reformulated.

Reconfiguring Costs and Planning Complexity
Since task scheduling involves accomplishing a number of actions that either consume assets or ties them for some periods of time both, scheduling complexity and re-planning cost vary greatly. Scheduling complexity depends on such factors as:

1. Precedence constraint or objectives prosecution order—represented in our formulation by chains of partial order sequences that establish not only the proper prosecution sequence for a Subtask but also the dependencies across subtasks. When these partial order chains indicate a cross sub-task dependency, the dependency becomes the basis for team coordination requirement. Typically, this will take the form of precedence and timing requirements that require coordination between teams executing different Subtask plans that would otherwise proceed without specific tactical awareness of each other. Factors such as delays and timing overlap must be resolved where inter Subtask dependencies exist.

2. Interference - A sub-task may interfere with the execution of another sub-task even when there are no clear inter-sub-task dependencies resulting from the partial order chains. There may some tactical disadvantage to execute a sub-task that triggers heightened enemy alert in a sector where another sub-task relies on surprise for example (overlap of a limited SEAD and a strike against near by TBMs to be more specific).
3. Resources conflict - It may not be possible to use a certain combination of resources during a specific interval or on the same sub-task.
4. Resource blank periods - A resource may be unavailable due to planned maintenance or due to planned use in another sub-task.

On the other hand, the cost for reconfiguring a platform and the planning complexity is based on the platform's state:

1. Bare platforms at bases (full capacity available for configuration) - no cost for reconfiguring, standard planning complexity
2. At base, pre-configured for another sub-task with:
   (a) partial capacity available for configuration - small cost for reconfiguring, standard planning complexity;
   (b) no capacity available for configuration - Moderate cost for reconfiguring, increased planning complexity (planning must expand to include the other sub-task).
3. Pre-configured platforms at position (x, y, z) that are
   (a) pre-configured, but not dedicated to an active task or sub-task - High cost for reconfiguring, moderate re planning complexity;
   (b) pre-configured for another active sub-task (resources are dedicated) - High cost for reconfiguring, highest re planning complexity.

Note: In most cases it is expected that it will be too costly to bring back to base a platform for reconfiguring.

4.2 Scheduling model formulation and solutions

At a minimum, the objectives for the scheduling process must be to maximize efficient use of resources while meeting mission goals. Additional objectives may be a preference for specific assets, minimizing unused weapon capacity or a deviation from some goal capacity, and minimizing overall time for task completion or deviation from some goal duration, to name a few.

There are additional constraints to consider such as an inventory of numerous weapon platform types with significantly different capabilities (endurance, range, weapon capacity/types, sensor capacity/types, etc.) and various weapon types consuming different amounts of the platform's weapon capacity.

Most scheduling problems can be formulated as a Linear Programming model and solved using classical methods. There is, however, no guarantee that a solution be reached. One way to overcome this problem is to redefine the

objectives or the constraints. This approach however, is not particularly suitable for automating the scheduling process. Instead, reformulation in the form a Goal Programming model enables us to convert most constraints to either an objective function or into a goal. The problem redefinition leads to a guaranteed solution that strives to minimize the deviation from the goals while meeting the objectives in the objective function. The addition of weights in the objective function further refines the model by prioritizing amongst objectives and leads to a better solution. This approach does not guarantee an optimal or an acceptable solution but it does give better solutions over the LP formulation by allowing better control through prioritization of the objectives and driving the solution in a desirable direction. The resulting confidence increase in the algorithm's solutions makes it suitable for use in automated scheduling. In cases where a solution requires violations of the goals, the human handler/operator is prompted for intervention and is expected to redefine the goals or change the list of task objectives.

Basic Assumptions Defining the Scheduling Problem
There are three basic assumptions related to the attempted scheduling process:

1. Resource are in limited supply;
2. tasks are composed of elementary parts called Operations and Delays;
3. Each task requires a certain amount of specified resources for a specific period of time called Process Time;
4. Resources are also made up of elementary parts: Machines, Cells, Transport, Delays.

The resources listed above are typical for the standard 'job-shop' scheduling problem. The scheduling objectives in our context are rephrased as follows.

Scheduling objectives
The scheduling objectives incorporated in our model formulation are to:

- Maximize efficient use of resources;
- Satisfy mission objectives;
- Minimize scheduling costs
    1. Positioning resources (transporting etc),
    2. Reconfiguration of resources (armament, sensors, and fuel),
    3. Delay costs resulting from timing considerations - holding cost additional resources (fuel and control).

Several methods can be used to find solutions to classical scheduling problems. Some of the most popular are solving for one objective at a time, solving for trade-off curve between objectives, and using weights with the later approach. In finding the solution to the Goal Programming model we combined objectives by assigning costs to desired outcomes (sub-task objectives) and to lack of resource utilization.

4.3 Team composition

Inventory
Limited resources require some efficient approach to allocation. In the context of the MICA project and other UAV strike missions it is assumed that any resource combination is likely to be of limited supply: quantity on UAV platforms of various types having significantly different capabilities (e.g. weapon and range), quantity of various weapon types, sensor types, and fuel. In the presented scheduling model formulation we limit the scheduling scope by considering only one limited resource, namely the quantity of three type of UAV weapon platforms and assume unlimited supply of weapons, fuel, sensors, etc. Let

$$\begin{aligned} p_1 &= \text{number of small\_weapon platforms} \\ p_2 &= \text{number of large\_weapon platforms} \\ p_3 &= \text{number of small\_combo platforms} \end{aligned}$$

Stated Goal
The primary and most significant goal of the process is to deliver to each of the sub-task's assigned targets a pre-determined weapon load. This is expressed as input in the form of a list of weapon type and number. The weapon association with each sub-task target is maintained externally and will be treated when it comes time to determine platform assignments. For the first primary goal we have:

$$\begin{aligned} \phi_1 &= \text{number of GPS bombs} \\ \phi_2 &= \text{number of Seeker Missiles} \\ \phi_3 &= \text{number of Munitions} \\ \phi_4 &= \text{number of Decoys} \end{aligned}$$

Selected Configuration
The solution that satisfies the sub-task constraints is a composition consisting of platform types,

$$\begin{aligned} \theta_1 &= \text{number of small\_weapon platforms} \\ \theta_2 &= \text{number of large\_weapon platforms} \\ \theta_3 &= \text{number of small\_combo platforms}, \end{aligned}$$

$$\text{Capacity } \alpha = \begin{cases} 10 & \text{for small\_weapon} \\ 20 & \text{for large\_weapon} \end{cases},$$

and weapon types,

$$\begin{aligned} \sigma_{1_i} &= \text{number of GPS bombs} \\ \sigma_{2_i} &= \text{number of Seeker Missiles} \\ \sigma_{3_i} &= \text{number of Munitions} \\ \sigma_{4_i} &= \text{number of Decoys} \end{aligned}$$

for $i = 1, \cdots, n$, in which n is the number of platforms.

4.4 Linear programming formulation

The task of allocating specific UAV platforms to specific targets can be formulated as a Linear Programming problem with multiple constraints. We first consider the desired outcome or objectives and define the following desired solution attributes.

The configuration (platform type and weapons) should:

1. Minimize need for additional inventory;
2. Maximize sub-task objective assignments;
3. Minimize unassigned capacity for each platform.

The decision variables are chosen to be:

$$\begin{aligned} X_1 &= \theta_1, \text{ the number of small\_weapon platforms} \\ X_2 &= \theta_2, \text{ the number of large\_weapon platforms} \\ X_3 &= \theta_3, \text{ the number of small\_combo platforms} \\ X_{k=j\times i+3} &= \sigma_{j_i}, j = 1, \cdots, 4; i = 1, \cdots, n; k = 4, \cdots, 4n+3, \\ &\quad \text{the number of seeker missiles, munitions, decoys for each platform} \end{aligned}$$

Constraints:
Available platforms:

$$\theta_i \leq p_i, \quad i = 1, 2, 3$$

Required weapons:

$$\begin{aligned} \phi_1 &\leq \sum_{i=1}^{n} \sigma_{1_i} = \text{GPS bombs} \\ \phi_2 &\leq \sum_{i=1}^{n} \sigma_{2_i} = \text{Seeker missiles} \\ \phi_3 &\leq \sum_{i=1}^{n} \sigma_{3_i} = \text{Munitions} \\ \phi_4 &\leq \sum_{i=1}^{n} \sigma_{4_i} = \text{Decoys} \end{aligned}$$

Available capacity: for $i = 1, \cdots, n$

$$\begin{aligned} \sigma_{j_i} &\leq \alpha_i, \quad j = 1, \cdots, 4 \\ \sigma_{1_i} + \frac{1}{2}\sigma_{2_i} + \sigma_{3_i} + \sigma_{4_i} &\leq \alpha_i \text{ or} \\ Cap_{\textit{remaining}_i} &= \alpha_i - \sigma_{i_1} - \frac{1}{2}\sigma_{2_i} - \sigma_{i_3} - \sigma_{i_4} \end{aligned}$$

Nonnegative constraints:

$$X_i \geq 0, \quad i = 1, \cdots, 4n + 3.$$

The equations:
Unassigned capacity: for $i = 1, \cdots n$

$$\begin{aligned} X_{4i+4} &\leq \alpha_i \\ X_{4i+5} &\leq 2 \times \alpha_i \\ X_{4i+6} &\leq \alpha_i \\ X_{4i+7} &\leq \alpha_i \\ X_{4i+4} + \frac{1}{2}X_{4i+5} + X_{4i+6} + X_{4i+7} &\leq \alpha_i \end{aligned}$$

Total number of platforms:

$$X_1 + X_2 + X_3 \leq p_1 + p_2 + p_3$$

Platform inventory usage:

$$X_i \leq p_i, \quad i = 1, 2, 3$$

Sub-Task objectives assigment:

$$\begin{aligned} \sum_{i=0}^{n-1} X_{4i+4} &\geq \phi_1 \\ \sum_{i=0}^{n-1} X_{4i+5} &\geq \phi_2 \\ \sum_{i=0}^{n-1} X_{4i+6} &\geq \phi_3 \\ \sum_{i=0}^{n-1} X_{4i+7} &\geq \phi_4 \end{aligned}$$

4.5 Goal programming formulation

Approach:

1. Use a weighting objective function model;
2. Reformulate constraints into Goals and place large weights on Goals that represent hard constraints (weapon capacity, available inventory, cross-platform assignments, etc).

The goals:
It is undesirable to:

1. Overachieve capacity for each platform $Cap_{\textit{remaining}}$—$\textit{Goal}_{l_i}$
2. Underachieve capacity for each platform $Cap_{\textit{remaining}}$—$\textit{Goal}_{l_i}$
3. Overachieve the total number of used platforms $\theta_1 + \theta_2 + \theta_3 \leq \textit{Goal}_2$—$\textit{Goal}_2$
4. Overachieve the platform inventory $\theta_1 - \theta_3$—$\textit{Goals}_3 - \textit{Goals}_5$
5. Underachieve the number of small_weapon platforms θ_1—$\textit{Goal}_3$
6. Underachieve sub-task objective assignments $\phi_1 - \phi_4$—$\textit{Goal}_6 - \textit{Goal}_9$

In summary,

$$
\begin{aligned}
\textit{Goal}_{l_i} &= \alpha_i \\
\textit{Goal}_2 &= \frac{\phi_1 + \phi_2 + \phi_3 + \phi_4}{\textit{large\_weapons}_{\textit{cap}}} = \frac{\phi_1 + \phi_2 + \phi_3 + \phi_4}{20} \\
\textit{Goal}_3 &= p_1 \\
\textit{Goal}_4 &= p_2 \\
\textit{Goal}_5 &= p_3 \\
\textit{Goal}_6 &= \phi_1 \\
\textit{Goal}_7 &= \phi_2 \\
\textit{Goal}_8 &= \phi_3 \\
\textit{Goal}_9 &= \phi_4
\end{aligned}
$$

4.6 Implementations in executable code

Synopsis
1) Sample task
2) Processing steps
3) Coupling with previous and following modules
Output and usage—The task plan, or solution of the scheduling process, is in the form of a list of platforms with a list of ordered assignments and a configuration appropriate for prosecution of all the assignments (weapons, sensors, and other payload). In the iteration process cost is evaluated for a solution based on threat exposure to platforms, efficient usage of resources, and on task completion (are all objectives assigned?).

Assembling the partial order sequences
assemble sub-task information cluster
1) Extract sequences
2) Get objective Info from “iad.xml” file
a. Add locations of entities
b. Identify objective type

Assessing required arsenal for task prosecution
Do weapon selection
1) Load weapon performance data
2) Input Desired Prosecution Effect
3) Match objective type and desired effect with best weapon performer

Team Composition and tasking Perform scheduling for the sub-tasks (given objective list, partial order of execution, weapons matching objectives) - Usage of Goal Programming to get platforms and configurations
1) Output team composition and tasking to a text file.

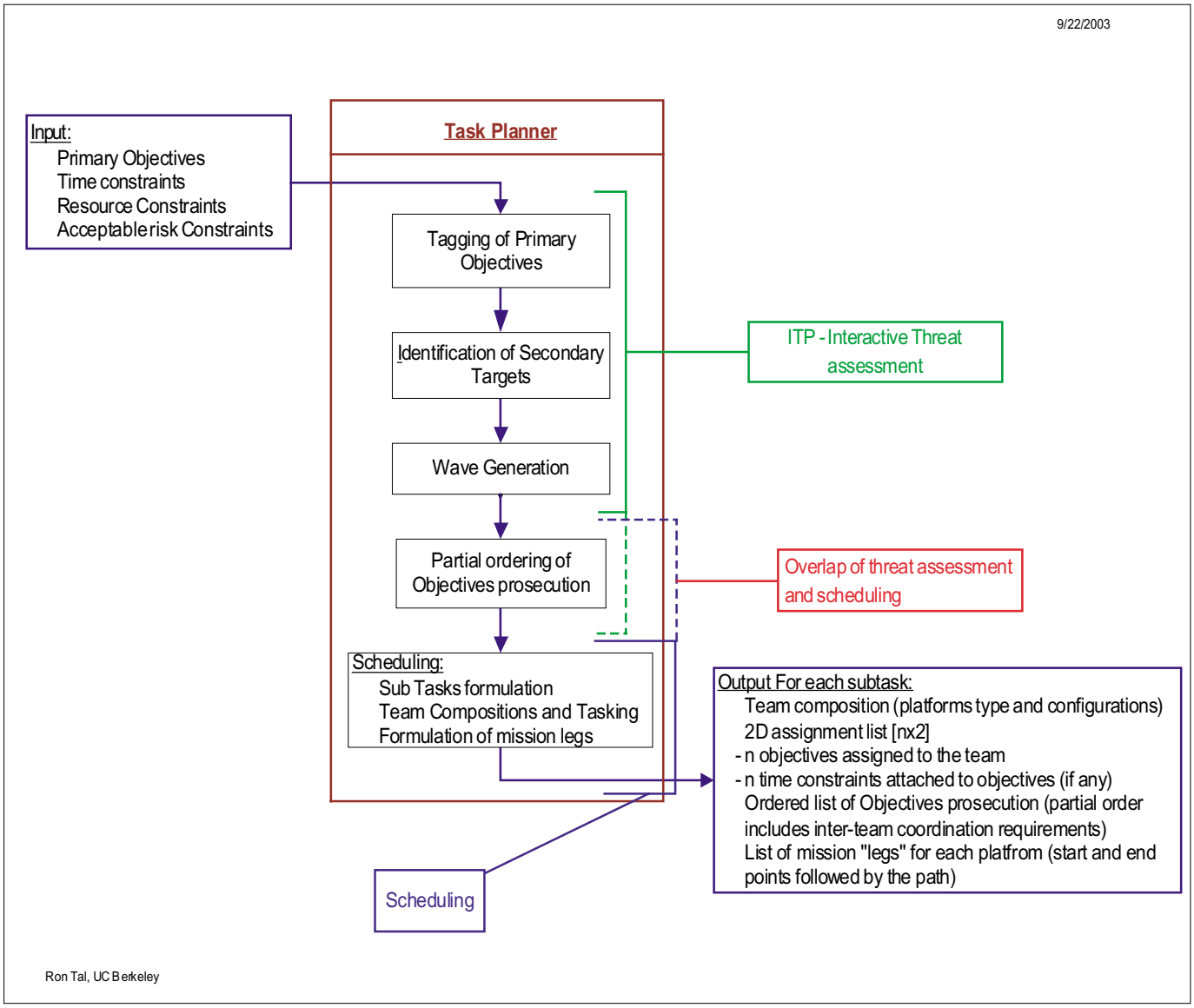


Figure 15: Data flow for module 2

5 Module 3: Task execution

5.1 Introduction

This section describes the control architecture and its implementation for task execution and teaming. The implementation is expressed in *Shift* and, unless stated otherwise, it will be called the 'execution environment'.

The execution environment allows the user to specify, execute, and supervise complex operations of multiple vehicles. Examples of such operations are tasks—executed by teams of UCAVs; missions—executed by isolated UCAVs; team formation and breakup; and vehicle-to-task or vehicle-to-mission re-allocation. This means that each vehicle may switch between two types of interactions: 1) independent operation during mission execution; and 2) team operation during task execution.

In this section we describe a specific implementation of the execution environment: it implements the control architecture, a library of vehicle maneuvers, and one specific task controller. The task controller implements the attack specifications generated by the ITP and by the configuration and schedule modules.

The execution environment models a complex distributed system, in which information and commands are exchanged among multiple vehicles, and the roles, relative positions, and dependencies of those vehicles change during operation. Therefore, dynamic reconfiguration is one of the key concepts in our execution control concept: we use a link not as a fixed part of the system but as a datum that we can manipulate.

We model the execution environment in the framework of dynamic networks of hybrid automata (DNHA)[10] Informally, DNHA allow for interacting automata to create and destroy links among themselves, and for the creation and destruction of automata. A hybrid automaton admits two types of interactions: 1) the differential inclusions, guards, jump and reset relations may be functions of variables from other automata, 2) automata may exchange events. The interactions are mediated by means of communication. The model for dynamic interactions includes a description of the mechanisms by which automata interact.[11] We adopt synchronous composition of hybrid automata, the underlying model of the *Shift* language.

Henceforth, and unless otherwise stated, we use the *Shift* terminology and notation to describe the execution environment. The exception consists in our use of 'messages' to describe communication among components. There is no message construct in *Shift*. We model synchronous message passing with other *Shift* constructs. However, and for the purpose of clarity, we use messages in our description of the execution environment. In this context messages

[10] A hybrid automaton consists of control locations or discrete states with edges or transitions between the control locations. The control locations are the vertices in a graph. A location is labelled with a differential inclusion, and every edge is labelled with a guard, and a jump and reset relation. Formally, a hybrid automaton is $H = (L, D, E)$ in which:

- L is a set of control locations.
- $D : L \rightarrow$ *Inclusions*, in which $D(l)$ is the differential inclusion at location l.
- $E \subseteq L \times$ *Guard* $\times$ *Jump* $\times L$ are the edges—an edge $e = (l, g, j, m) \in E$ is an edge from location l to m with guard g and jump relation j.

The state of a Hybrid Automaton is a pair (l, x) where l is the control location and $x \in R^n$ is the continuous state.

[11] At the level of software implementation the mechanisms by which software modules interact are called models of computation, or semantic frameworks.

are typed events. Commands are encoded as messages. For each component there are two types of events: input and output events (*in* and *out*).

5.2 An aside on *Shift*

Shift is a specification language for describing networks of hybrid automata. *Shift* users define types (classes) with continuous and discrete behavior as depicted in figure 16. A simulation starts with an initial set of components that are instantiations of these types. A component is an input-output hybrid automaton. The world-evolution is derived from the behavior of these components. A type consists of numerical variables, link variables, a set of discrete states, and a set of event labels—together, these constitute a description of the data model. The variables are grouped into input, state, and output variables.

```
type Vehicle  {
     input      (what we feed to it)
     output     (what we see on the outside)
     state      (whats internal)
     discrete   (discrete modes of behavior)
     export     (event labels seen from the outside)
     flow       (continuous evolution)
     transition (discrete evolution)
     setup      (actions executed at create time)
 }
```

The inputs and outputs of different components can be interconnected. Each discrete state has a set of differential equations and algebraic definitions (flow equations) that govern the continuous evolution of numeric variables. These equations are based on numeric variables of this type and outputs of other types accessible through link variables.

The transition structure of the hybrid automaton may involve synchronization of pairs or sets of components. The system alternates between the continuous mode, during which the evolution is governed by the flow equations, and the discrete mode, when simulation time is stopped and all possible transitions are taken, as determined by guards and/or by event synchronizations among components. During a discrete step components can be created, interconnected, and destroyed. The continuous mode is implemented by a fixed step Runge-Kutta integration algorithm and the step size determines the accuracy of the simulation. *Shift* allows hybrid automata to interact through dynamically reconfigurable input/output connections and synchronous composition. The first order predicate constructs of *Shift* (e.g. existential and universal quantification) are used to provide compact representations of dynamic synchronous composition.

Notation

In what follows, and for the purpose of clarity, we define input and output events in the transitions of a component. Input events originate in a different component, and output events are generated by the component itself. An event labelled e originating from a component called c is denoted by $c : e$. Another component linked to c access the output variable v of c as $v(c)$.

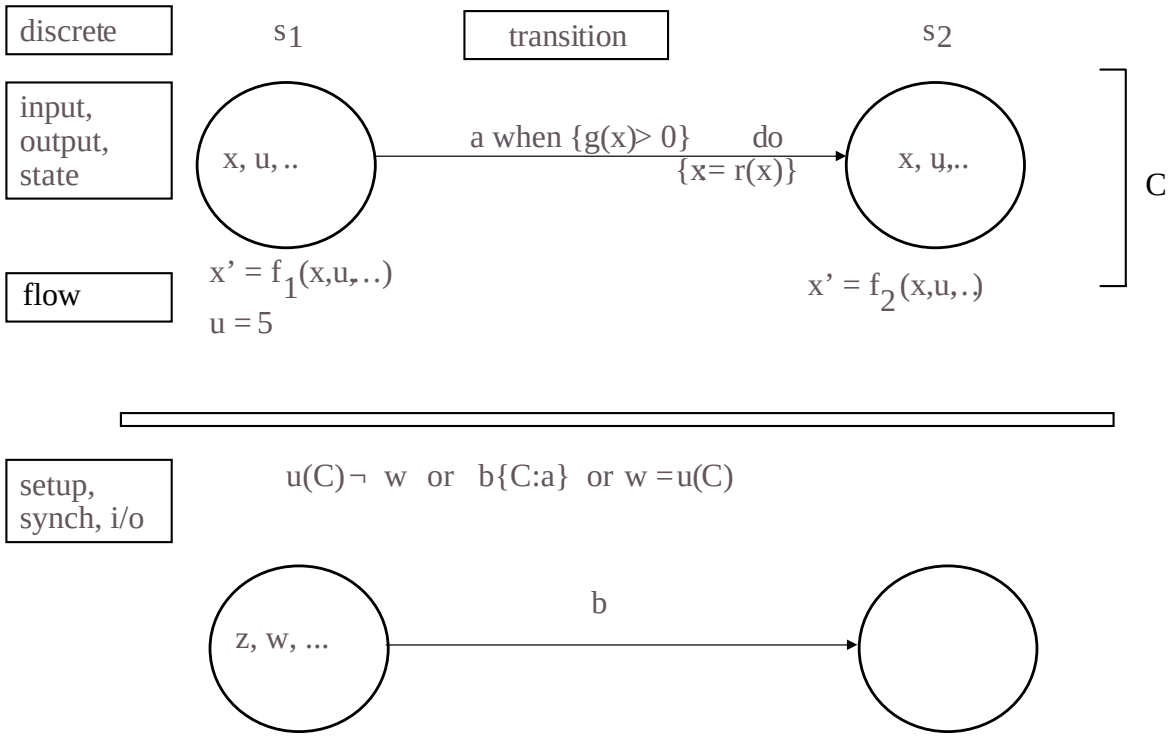


Figure 16: Shift – dynamic networks of hybrid automata.

5.3 Architecture

We adopted the following design principles:

- Separation of specification and controller implementation: each controller accepts specifications conforming to a specification format. There are different controller implementations for the same specification. In this case, it is up to the user to specify both the specification and the type of controller to execute it;
- Separation of controller implementation from their localization: each controller may reside onboard a vehicle or on a remote location, and it can be moved from one location to another;
- Layers of control and abstraction: controllers are layered and described within a particular theory for each layer;

- Independence of the cardinality of teams of vehicles: the structure of multi-vehicle controllers is independent of the number of vehicles controlled;
- Layers of user intervention: modes of user intervention are available at all layers of control for mixed initiative operation;
- Extensibility: additional layers of control and abstraction can be installed on top of existing ones.

The design uses the following specification concepts:

Maneuver: a prototype of an action/motion description for a single UCAV and the atomic component of all specification concepts.

Mission: an array of maneuvers to be executed sequentially. Other mission structures are possible, but are not implemented.

Task: a prototype of an action/motion description for a group of vehicles (team).

To each type of specification concept there corresponds one type of controller. There are two types of controllers: vehicle controllers and team controllers. Vehicle controllers control mission and maneuver execution. Team controllers control task execution.

The implementation of the control architecture, depicted in figure 17 uses these key concepts:

Platform. A 'mirror' of the OEP platform and the interface to the OEP. It mirrors the state of the platform on the OEP and accepts commands for weapons, sensors, and ESM devices from maneuver controllers. It sends commands to and receives data from the OEP.

Maneuver controller. Supervises the execution of a vehicle maneuver. It sends commands to the *platform* and gets the current status from it. It accepts abort and configuration commands from the *vehicle supervisor* and sends status messages to it. It is created by the *vehicle supervisor* and it deletes itself when done. There is always one active maneuver controller in the platform.

Vehicle supervisor. Supervises all of the UCAV operations. It receives maneuver specifications through a link to either the *dispatcher* during a mission execution ,or to a *team controller* during a team task execution, and launches the corresponding maneuver controller and monitors its execution and the state of the vehicle, and accepts configuration commands from an external controller. For example, it is possible to change the link to a team controller. This means that it is possible to move the vehicle among teams. The vehicle supervisor is the same throughout the life span of the UCAV. If there is no link to a team controller and no mission to be executed the *vehicle supervisor* commands the execution of a default maneuver, typically *goto* base.

Vehicle dispatcher. Supervises the execution of a *mission*. Basically repeats the following pattern of interactions: Gets the next maneuver specification from the *mission*, sends it to the *vehicle supervisor* for execution, and waits for its completion.

UCAV. The UCAV unit. It is composed of a *platform*, the *vehicle supervisor*, and the *dispatcher*. It interacts with the OEP through the *platform* component, and with external entities, such as a *team controller* through the *vehicle*

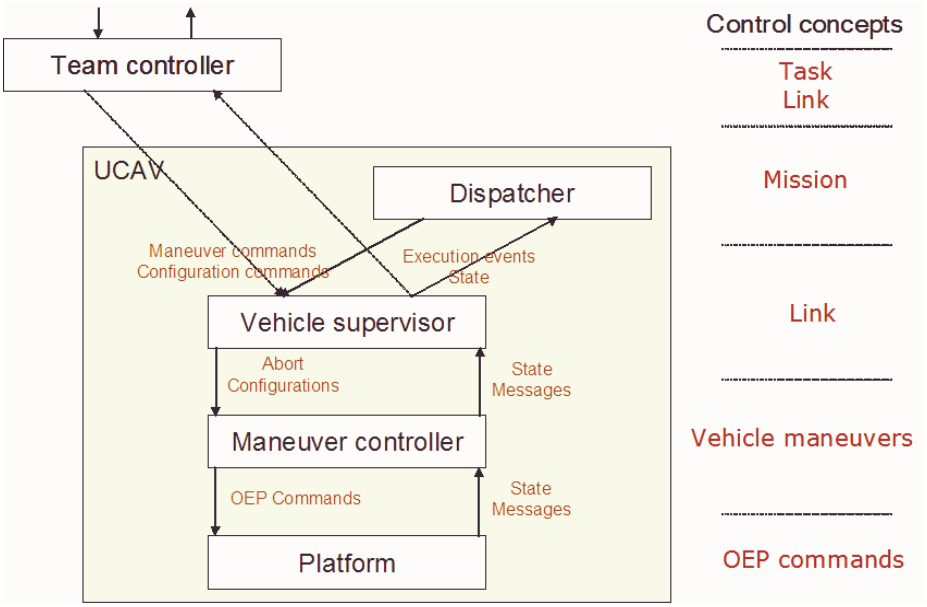


Figure 17: Control architecture.

supervisor

Team controller. Supervises the execution of a task. Basically it commands and monitors the execution of vehicle maneuvers to execute the task specification. It does this by exchanging messages with the *vehicle supervisors* in the team: it uses the protocol that governs the interactions between the *vehicle supervisor* and the *dispatcher*. It also provides for a definition of a team—as a set of UCAVs under the control of a team controller. The team controller, in turn, may be composed of several different controllers. The team controller also accepts configuration and task execution and abort commands. This allows for interactions with higher layers in the architecture, not depicted in figure 17.

The architecture allows for incremental development. There is a vehicle maneuver library and a task library. The current implementation includes several vehicle maneuvers and one task.

Next, we describe the controllers and the main components in more detail. We used the inheritance constructs from *Shift* to define a hierarchy of maneuver specifications and of maneuver controllers. There is a base type for each *maneuver specification*, *maneuver controller* and *team controller*.

5.4 Mixed initiative interactions

The **mixed initiative** interactions are described next:

- Specify and command the execution of tasks and missions.
- Interrupt task or mission execution for task or mission re-planning.
- Change current task and mission configurations. This includes: 1) the addition and/or removal of vehicles from teams; 2) moving controller locations.

Figure 18 describes how an operator can command execution of manueuvers. The operator commands permit (1) maneuver selection, including remotely piloted operation (manual control); (2) creation of a link; (3) aborting execution; and (4) configuration.

Figure 19 describes the architecture of a team controller. Here, too, a human operator (not shown) can take over command of the team operation by issuing commands that create/delete or configure a team supervisor.

5.5 UCAV type

The *UCAV* type describes the UCAV data model. It defines internal links – with respect to the UCAV – and external links – with respect to team controllers. These are made available as output variables so that other components can access them through a link to the *UCAV*.

Output variables		
variable	***type***	***description***
d	dispatcher	link to platform dispatcher
vs	vehicle_supervisor	link to vehicle supervisor
p	platform	link to OEP platform
tc	team_controller	link to team controller

5.6 Platform type

The platform type basically interfaces the execution environment to the OEP. The interface consists of ‘read-only’ or ‘read-write’ variables and commands. The interactions with the OEP occur at each OEP time step: the OEP stops execution, the ‘execution environment’ runs and the *platform* sends commands to the OEP and, finally, the execution environment commands the OEP to advance another time step.

Output variables		
variable	***type***	***description***
x,y,z	number	platform position from the OEP
fuelRate	number	fuel consumption rate (read-write)
fuelRemaining	number	quantity of fuel on-board (read-only)
bombsRemaining	number	number of GPS bombs on-board (read-only)
damageStatus	number	status (read-only) 0 unknown, 1 undamaged, 2 damaged, 3 destroyed
speed	number	platform speed (read-write)
destx, desty, destz	number	platform destination (read-write)
tx,ty,tz	number	target location (read-write)
ex,ey,ez	number	emitter direction location (read-write)
sensorLocksDetected	number	counter of sensor locks (read-only)
t	number	timer (read-write)

Exported events		
event	***description***	***to/from***
setDestination (out)	set destination to *(destx, desty, destz)*	OEP
setEmitterOn (out)	jam location *(ex, ey, ez)*	OEP
setEmitterOff (out)	set jammer off	OEP
activateWeapon (out)	launch weapon to target location *(tx, ty, tz)*	OEP
ready (out)		
destroyed (out)		

Discrete states	
state	***description***
init	OEP initialization procedures
operational	normal platform operation
engage	waits one time step to launch one bomb
inoperational	platform destroyed
error	OEP error
exit	state where the component is deleted

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
init	operational	OEP platform ready		ready	
operational	operational			setEmitterOn	jam location *ex, ey, ez*
operational	operational			setEmitterOff	stop jamming
operational	operational			setDestination	set destination to *destx, desty, destz*
operational	engage	$bombsRemaining > 0$		activateWeapon	reset timer t:=0
engage	operational	$t > 0$		activateWeapon	
operational	inoperational	damageStatus=3		destroyed	

5.7 Maneuver specification

A *maneuver controller* takes as an input a *maneuver specification*. The *maneuver specification* type is basically a data model which encodes the parameters required by the controller to execute a maneuver.

5.7.1 Base type

The base type defines a data model shared by all maneuver specifications.

Output variables		
variable	***type***	***description***
speed	number	platform speed
accuracy	number	way-point tracking accuracy
mintime	number	minimum execution time
maxtime	number	maximum execution time
typ	symbol	type of maneuver

5.7.2 Types of maneuvers

There are specializations for each type of maneuver specification. The types of maneuvers available in the current implementation are:

holding – the UCAV flies a holding pattern. In the current implementation the holding pattern is a rectangle. The parameters of the maneuver are : 1) length and height of the rectangle; 2) maximum duration.

attack_jam – jams and attacks a SAM site with 2 GPS bombs. The attack path presents the minimum radar signature to a radar co-located with the SAM site. The jammer is activated when the vehicle is within jamming range of the target. The bombs are released when the target is within weapons range.

goto – go from the current location to a given location.

jam_site – go from the current location to a given location and jam another location.

follow_path – follow a given path with a certain speed profile.

5.7.3 Example: attack_jam

The **attack_jam** *maneuver specification* inherits the data model from the base maneuver specification and adds the following outputs.

Output variables		
variable	***type***	***description***
destx, desty, destz	number	final destination
ex, ey, ez	number	location of emitter to be jammed
tx, ty, tz	number	target location
weaponsRange	number	weapons release distance
weapon	symbol	type of weapon
jammingRange	number	effective jamming radius
target	platform	platform to be attacked

A mission to attack a target may involve spatial coordination of two or more UAVs. If one UAV reaches its destination earlier than another, the team controller specification may trigger the **hold** maneuver.

5.8 Maneuver controller

5.8.1 Base type

The base maneuver controller type defines the data model shared by all maneuver controllers.

Output variables		
variable	***type***	***description***
v	platform	link to platform
vs	vehicle supervisor	link to vehicle supervisor

Exported events		
event	***description***	***to/from***
stop	maneuver is done (out)	vs
e_nobombs	platform runs out of weapons (out)	vs
init_not_ok	initialization error (out)	vs

Discrete states	
state	***description***
initialize	maneuver initialization procedures
execution	normal execution state
error	error state
done	successful termination procedures
exit	state where the component is deleted

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
all	exit		v:destroyed		remove self
all	exit		vs:abort	abort	remove self

There are specializations for each type of maneuver controller, one per type of maneuver specification.

5.8.2 Example: attack_jam type

The *attack_jam* maneuver controller type adds discrete states and input messages to the base component, and defines the discrete and continuous evolution.

Input messages
execute(m_specification)

Output messages
goto(x,y,z)

Discrete states	
state	***description***
beginInfil	transition state
lowProfileInfil	moves to the point where it starts jamming
jammingInfil	jams and moves to the point where it releases a GPS bomb
engage	waits one time step to release a second bomb
beginExfil	transition state

The controller basically sends commands to the *platform*. Note that the column 'action' refers to actions in the *Shift* environment.

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
initialize	beginInfil	true	execute(m_specification)	v:goto(x,y,z)	
beginInfil	lowProfileInfil	true		v:goto(x,y,z)	
lowProfileInfil	jammingInfil	within jamming range		v:setEmitterOn	
jammingInfil	engage	within weapons range		v:activateWeapon	
engage	beginExfil			v:activateWeapon	
beginExfil	exit			vs:stop	remove self

5.9 Vehicle supervisor

The *vehicle supervisor* supervises and controls the UCAV. It accepts commands from either the *dispatcher* or from a *team controller* when the flag *accept* is set to 1.

Output variables		
variable	***type***	***description***
mc	maneuver_controller	link to current maneuver controller
ms	vehicle_maneuver	link to current maneuver specification
accept	number	flag to accept abort commands
u	UCAV	link to the UCAV where it resides
v	platform	link to platform

Exported events		
event	description	to/from
abort	abort command (in)	mc
destroyed	vehicle destroyed (out)	u
done	maneuver terminated (out)	u
e_nobombs	run out of bombs (out)	u
e_maneuver_init_fails	initialization failed (out)	u

Input messages
execute(m_specification)

Discrete states	
state	***description***
initialize	maneuver initialization procedures
execution	normal execution state
idle	waiting for commands (transition state)
error	error state

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
initialize	idle		p(u):ready		accept:=0
idle	execution	tc(u)=nil	execute(m_specification)		create(maneuver_controller)
idle	execution	tc(u)/=nil	execute(m_specification)		create(maneuver_controller)
execution	idle		mc:stop	done	
execution	idle		mc:e_nobombs	e_nobombs	
execution	idle		mc:init_not_ok	e_maneuver_init_fails	
execution	idle	accept=1		abort	
idle	execution	d(u)=nil and tc(u)=nil			create(maneuver_controller)
all	exit		p:destroyed	destroyed	remove self

5.10 Vehicle dispatcher

5.10.1 Mission specification

The *dispatcher* supervises the execution of a *mission.* In the current implementation the data model of a mission is described by the type *mission* which consists of an array of maneuver specifications: $mp = [mspec_1, \ldots, mspec_n]$.

Output variables		
variable	***type***	***description***
step	number	number of maneuver specifications
mission	array(mspec)	array of maneuver specifications

5.10.2 Dispatcher type

The *dispatcher* maintains the state of the execution of the *mission* – the index of the last maneuver executed successfully – and when it receives the *done* event from the *vehicle supervisor* it increments the index by one and commands the *vehicle supervisor* to execute the next maneuver specification. This is only possible when the *accept* flag is set to 1.

Input variables		
variable	***type***	***description***
mp	mission	mission plan
vs	vehicle supervisor	link to vehicle supervisor
m	maneuver specification	specification of current maneuver
accept	number	flag to accept commands

Output variables		
variable	***type***	***description***
c	number	index of last maneuver executed
n	number	number of maneuvers in mission plan

Out messages
execute(m_specification)

Discrete states	
state	***description***
initialize	maneuver initialization procedures
execution	normal execution state
idle	waiting for commands (transition state)
error	error state

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
initialize	idle	accept=1			c:=0
idle	execution	$c < n$		execute(m_specification)	c:=c+1
execution	idle		vs:abort		
execution	idle		vs:done		
all	error		vs:e_nobombs		

5.11 Task specification

5.11.1 Concepts

Here, we describe the type of task specification generated by the ITP and the configuration and schedule modules. First, some definitions.

Leg: a specification for a sequence of two maneuvers with two alternative modes of execution. The goal of the first mode consists of destroying a target by releasing 2 GPS bombs at the end of a sequence of two concatenated paths. The sequence is designed to minimize risk. The goal of the second mode consists of reaching the end point of the two paths. The mode of execution of the *leg* depends on the state of the target. If the target has not been destroyed then the leg consists of a *follow_path* maneuver followed by a *attack_jam* maneuver. The first maneuver consists in tracking a given path, which is assumed to be safe at the planning stage. The path may be empty. The final point of this path is the starting point of the second maneuver, which consists of flying a minimum radar signature path to release weapons at the prescribed target while jamming a radar location. Weapons release takes place when the target is within a pre-specified range, typically, the weapons' range. If the target has been destroyed the leg consists of two consecutive *follow_path* maneuvers. The second one replaces the *attack_jam* maneuver. For example, in figure 20, the paths corresponding to the two parts (or maneuvers) of the leg consist of two straight lines. The first one is the safe path and the second one is the attack path.

Task: a set of legs together with a partial order for their execution. The task specification encodes the wave structure plus the configuration and schedule components from the planner specification. The task is organized as a set of sub-tasks that are to be executed concurrently with execution dependencies.

Sub-task: a sequence of legs satisfying a total order induced by the task partial order.

Formally, a *Task specification* is a pair:

$task = \{(SubtaskList, \succeq), (TeamList, assign)\}$ where:

- $SubtaskList = \{subtask_1, \ldots, subtask_n\}$ is a set of sub-tasks, each of which is an array of legs, $subtask_i = \{leg_{i,1}, \ldots, leg_{i,in}\}$. There is a partial order $\succeq$ on the legs composing a task. The legs composing a sub-task satisfy a total order. However, there is a partial among legs on different sub-tasks.
- $TeamList = \{team_1, \ldots, team_n\}$, and $\{assign : TeamList \rightarrow SubtaskList\}$ is an assignment (or 1-1 function) of teams to sub-tasks.

5.11.2 Leg type

The *leg* type describes the data model for leg specification. It consists basically of output variables.

Output variables		
variable	***type***	***description***
p_attack	array(array(number))	safe path segment of the leg
aspeed	number	attack speed
hold	maneuver specification	specification for holding maneuver
t	platform	target for attack phase
r	set(leg)	preceding legs

5.11.3 Subtask type

Figure 21 describes a task specification. It basically consists of *sub-task specifications* and a partial order on the execution of the constituent legs. The data model of a *subtask* is described by the type *subtask* which consists basically of output variables.

Output variables		
variable	***type***	***description***
p	array(leg)	sequence of legs composing the sub-task
steps	number	number of legs
team	set(ucav)	team assigned to execute the sub-task

5.12 Team controller

5.12.1 Base type

The base type just outputs a timer and the next maneuver specification to be sent to a vehicle supervisor, and exports the event 'abort'.

5.12.2 Task controller

The *task controller* type inherits from the type *team controller*. It takes as an input a *task specification* and creates one *subtask controller* for each *sub-task* in the specification. It maintains one link to each *sub-task controller*, a list of the legs executed so far and its own state of execution (normal or fail).

Input variables		
variable	***type***	***description***
t	task	task specification
task_legs_done	set(legs)	legs executed so far (coordination variable)
fail	number	fail flag (coordination variables)

Input variables		
variable	***type***	***description***
st	set(sub_task_controller)	links to all sub-task controllers

The **coordination variables** are *task_legs_done* and *fail*. Each *subtask controller* reads the first variable to conform to the execution partial order. In the current implementation, when at least one of the *sub-task controller* fails, it sets the *fail* variable of the *task controller* to 1 (fail) and all *sub-task_controller* enter a fail mode where the corresponding UCAVs are commanded to move to the closest safest point and enter a holding pattern when they arrive there.

5.12.3 Sub-task controller

The *sub-task controller* takes as an input a *subtask specification* and controls and coordinates the concurrent operations of two sets of vehicles: *attackers*, *reserve*. To do this, it maintains the state of execution of each set of vehicles, the state of execution of the sub-task, and keeps two sets of vehicle supervisors, *vsa* and *vsr*, for the *attackers* and for the *reserve* sets respectively. In the current implementation, the set *attackers* consists of just one vehicle, when it is not empty. The transition structure of the *sub-task controller* hybrid automaton is defined on the states of the two sets of vehicles and on the state of execution of the *sub-task*. This provides for a high level of abstraction and for a more compact notation.

The state of each set of vehicles is described by the current leg, the index of the current leg, the constituent vehicles of the set itself, and the execution state.

State of *attackers:*

current_leg: current attack leg.

a_step: index of the *current_attack_leg* in the *sub-task*.

attackers: set of attackers.

Execution state: (one of the following)

$attack – executing *attack_jam* segment of *current_attack_leg*.

$path – executing *follow_path* segment of *current_attack_leg*.

$hold – executing *holding* maneuver waiting for some other leg in the *task* to be executed.

$nil – *attackers* has not been created yet.

State of *reserve*:

Current leg: The current reserve leg is maintained by variable *current_reserve_leg*.

r_step: The index of the *current_reserve_leg* in the *subtask*.

reserve: Set of reserve vehicles.

Execution state: (one of the following)

$hold_end – all of the reserve vehicles are executing holding maneuver at the end of *current_reserve_leg*.

$hold_path – all of the reserve vehicles are executing holding maneuver at the end of the first part of *current_reserve_leg*.

$path – at least of one reserve vehicle is still executing a *follow_path* – maneuver in the first part of *current_reserve_leg*. The others are already executing a *holding* maneuver.

$path_attack – at least of one reserve vehicle is still executing a *follow_path* maneuver in the final part of *current_reserve_leg*. The others are already executing a *holding* maneuver.

The actions on the transition system consist in one of the following: 1) command maneuver execution to vehicle supervisors in *vsa* or/and in *vsr*; 2) transfer vehicles from the *reserve* to the *attackers* set when the second set becomes empty; 3) remove a vehicle from *reserve* and/or *attacker* when the vehicle is destroyed or it has to leave the team for some other reason. The set *attackers* is empty when the *attackers* vehicle is destroyed or runs out of bombs. In the last case, the corresponding UCAV leaves the set and the team executing the *subtask*.

The control logic is briefly described next. Initially, the *attackers* set is empty while the *reserve* set receives the team of vehicles allocated to the sub-task controller. Execution starts with the first leg of the *subtask*. The *reserve* vehicles execute a *follow_path* maneuver to follow the safe path that composes the first path of this leg. When the end of that path is reached, one of the reserve vehicles is transferred to the *attackers* set and the two sets of vehicles start two concurrent threads of execution until the *subtask* terminates successfully or fails. The *attackers* and *reserve* execute each leg differently, as described before.

The *attackers* vehicle leads the execution. It executes the *sub-task specification* until successful termination, or until it is destroyed or runs out of bombs. The *reserve* set stays behind the *attackers* in terms of the execution of the *subtask*. Basically, it advances to the farthest safe point in the *subtask* legs executed so far. This points moves forward as the *attackers* destroy targets. The *reserve* team just follows the paths defined for each leg.

The user may change the execution logic. For example, when the *attackers* vehicle is destroyed the user may require the *sub-task controller* to enter a fail mode. In this mode, the *sub-task controller* that failed sets the *fail* flag in the corresponding *task controller*. Then, all the *sub-task controllers* enter the fail mode. In this mode, the *reserve* team is required to hold in place and the *attackers* are required to rendezvous with the *reserve* set only after completion of the risky parts of the *subtask*. In practice, this means that it will keep advancing until it reaches a safe region. The current implementation accommodates both the case above described and the case where the *attackers* vehicle is destroyed and one of the reserve vehicles is allocated to replace it.

The *Shift* encoding of this controller follows. We will skip some of the implementation details for the sake of clarity.

Input variables		
variable	***type***	***description***
st	subtask	link to the sub-task specification
tc	task controller	link to the task controller

The output variables are used as **coordination variables** between *attackers* and *reserve*.

Output variables		
variable	***type***	***description***
reserve	set(UCAV)	set of reserve vehicles
reserve_hold	set(UCAV)	set of reserve vehicles in holding positions
attackers	set(UCAV)	set of attack vehicles (one)
current_reserve_leg	leg	leg being executed by reserve vehicles
current_attack_leg	leg	leg being executed by attack vehicles
r_step	number	index of current_reserve_leg in the *sub-task*
a_step	number	index of current_attack_leg in the *sub-task*
vsa	set(supervisors(attackers))	set of vehicle supervisors of attackers
vsr	set(supervisors(reserve))	set of vehicle supervisors of reserve
attack_stage	symbol	state of *attackers* execution ($attack, $path, $hold)
reserve_stage	symbol	state of *reserve* execution ($hold_end, $hold_path, $path, $path_attack)
team	set(UCAV)	team executing this sub-task

Exported events		
event	description	to/from
abort	abort command (in)	tc
init_not_ok	error event (out)	tc

There is basically one state *execution* where the normal operation of the controller takes place. The other states are either preparation states (*initialize, enroute_rendezvous*) or error or fail states.

Discrete states	
state	***description***
initialize	transition state for initialization procedures
enroute_rendezvous	vehicles in the first path of the first leg
execution	normal execution state
fail	failing mode of execution
error	error state
success	(transition state to exit)
exit	state where the component is deleted

There a partition of the transition structure. The majority of the transitions are self-loops for each discrete state, in particular for the *execution* state. The other transitions concern transitions among the discrete states. The self-loops have guards on the state of execution of each set of vehicles, and have in/out events and in/out commands exchanged with corresponding sets of vehicle supervisors, *vsa, vsr*.

Mathematically, this is equivalent to defining the transition structure on a larger set of discrete states. However, this style is more convenient for programming since the behaviors of *attackers* and *reserve* can be specified independently (with some coupling resulting from precedence relations on attack and reserve legs) and transitions can be easily added and deleted without additional changes on the remaining code. This style also allows for a more convenient abstraction of the coordination structure and can be easily extended to more sets of vehicles, for example vehicles in charge of battle damage assessment.

The transition structure involving transitions among the different discrete states is briefly described next.

Transition					
from	***to***	***condition***	***input event***	***output event***	***action***
initialize	error	reserve={} or subtask=[]		init_not_ok	remove self
initialize	enroute_rendezvous	reserve/={} and subtask=/[]		vsr:execute(maneuver)(all)[12]	
enroute_rendezvous	execution	size(reserve_hold)=size(reserve)-1	vsr:done(one:p)[13]		
execution	fail	fail(tc)=1			
execution	fail	vsr:destroyed			fail(tc):=1
execution	success	reserve={} and a_step = size(p(st))	vsa:done(one:p)		remove self

The self-loops concerning the *execution* discrete state are briefly described next.

1. When reserve_stage=$holding_path and *attackers*={} allocate one *reserve* vehicle to *attackers* and start attack phase of the *current_a_leg*.

condition	***input event***	***output event***	***action***
1			1

2. When reserve_stage=$holding_end and *attackers*={} is empty and there are no precedences for the next attack leg allocate one *reserve* vehicle to *attackers* and start path phase of that leg.

3. When *attackers* finish executing the *follow_path* maneuver of *current_a_leg* command it to execute the *attack_jam* maneuver of that leg.

4. When *attackers* finish executing the *attack_jam* maneuver of *current_a_leg* command it to update *current_a_leg* to the next leg in the *subtask* and to execute the *follow_path* maneuver of that leg if there are no precedences.

5. When *attackers* finish executing the *attack_jam* maneuver of *current_a_leg* command it to execute a *holding* maneuver if there are leg precedences.

6. When attack_stage=$hold finish and leg precedences have been removed update *current_a_leg* to the next leg in the *subtask* and command *attackers* to execute the *follow_path* maneuver of that leg.

7. When reserve_stage=$moving_end and one of the *reserve* vehicles has finished the corresponding *follow_path* maneuver and $size(reserve\_hold/ = size(reserve))$ command it to execute a *holding* maneuver to wait for the other *reserve vehicles*.

8. When reserve_stage=$moving_end and the last *reserve* vehicle has finished the corresponding *follow_path* maneuver ($size(reserve_hold/ = size(reserve) - 1)$ command it to execute a *holding* maneuver.

9. When reserve_stage=$holding_end and $a\_stage > r\_stage$ and there is one p_attack/=[] in the legs between *current_r_leg* and *current_a_leg* (it is safe to move forward) then update *current_r_leg* to the next leg in the *subtask* and command the *reserve* vehicles to execute the corresponding *follow_path* maneuver.

10. When reserve_stage=$moving_path and $size(hold\_reserve) < size(reserve) - 1$ and there is one *reserve* vehicle that has just completed the *follow_path* maneuver command it to execute a *holding* maneuver to wait for the other vehicles.

11. When reserve_stage=$moving_path and $size(hold\_reserve) = size(reserve) - 1$ and the last *reserve* vehicle has just completed the *follow_path* maneuver command it to execute a *holding* maneuver and update reserve_stage=$holding_path.

5.12.4 Properties

1. In the case of perfect information the *reserve* team always flies a safe path.
2. The *attackers* fly a path that minimizes the maximum risk.

5.13 Conclusion

The control architecture and the *Shift* implementation allow for additional layers on the top of the existing ones, and also for the extension of the libraries of maneuver and team controllers. For example, the implementation of the following control layers on the top of the existing ones is straightforward:

- Transfer of vehicles among tasks, and between base and tasks.
- Dynamic team re-allocation.

The structure of the task controller encodes a framework for multi-team coordination and control by defining a transition structure on variables describing the state of each group of vehicles thus providing for a convenient state aggregation.

The structure of the task specification allows for a compact representation that is interpreted differently according to the state of execution and to the type of abstraction utilized. For example, the execution of a leg depends on the type of vehicles engaged in executing it.

The task controller can be extended to accommodate other types of objective functions, for example, to minimize the time to get to final target while maintaining the properties described above.

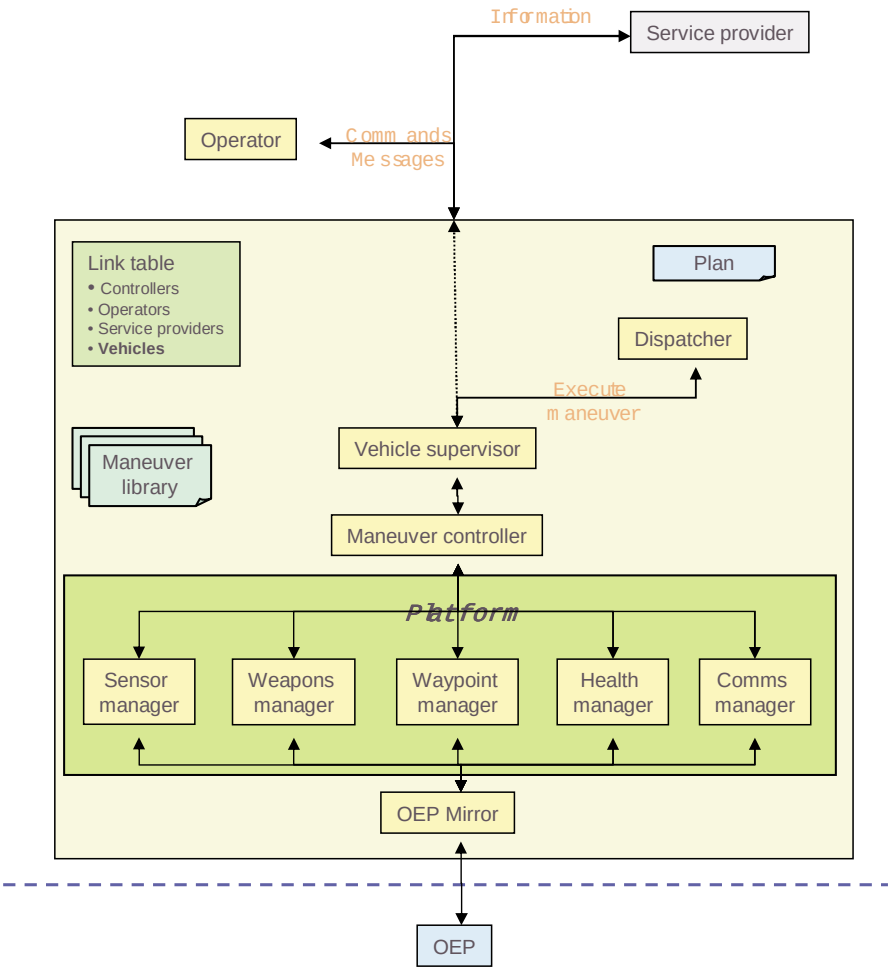


Figure 18: Individual mission execution.

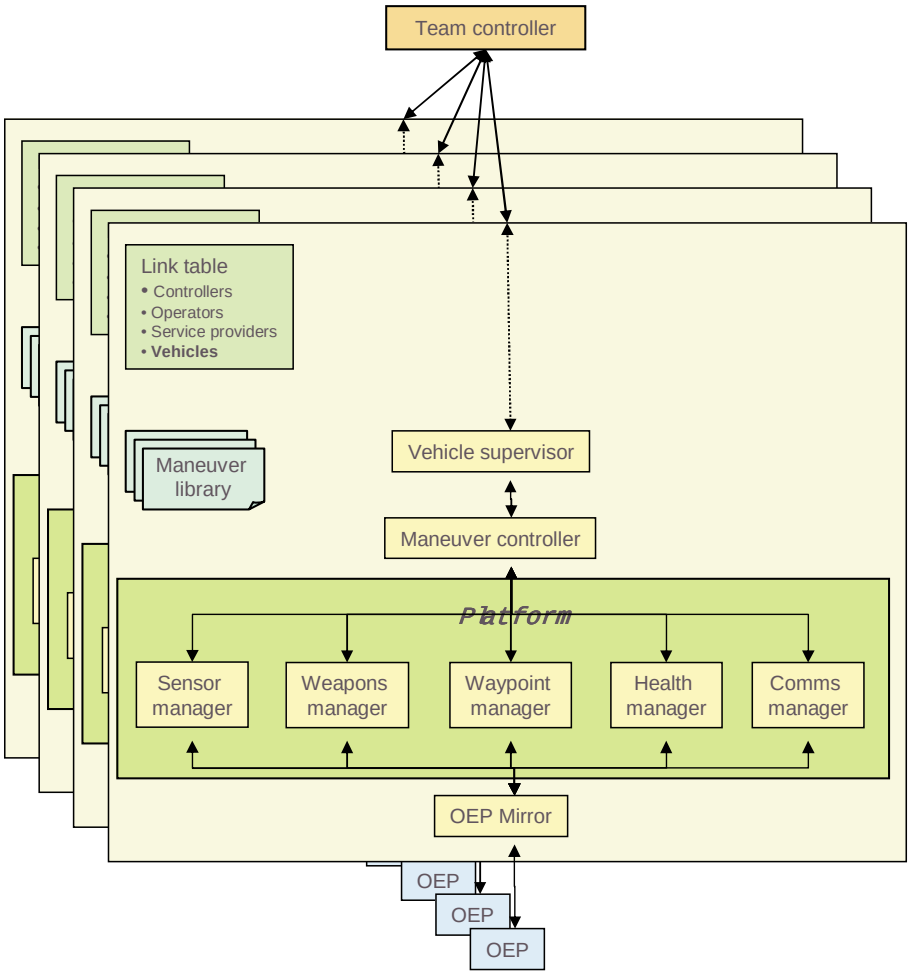


Figure 19: Team control.

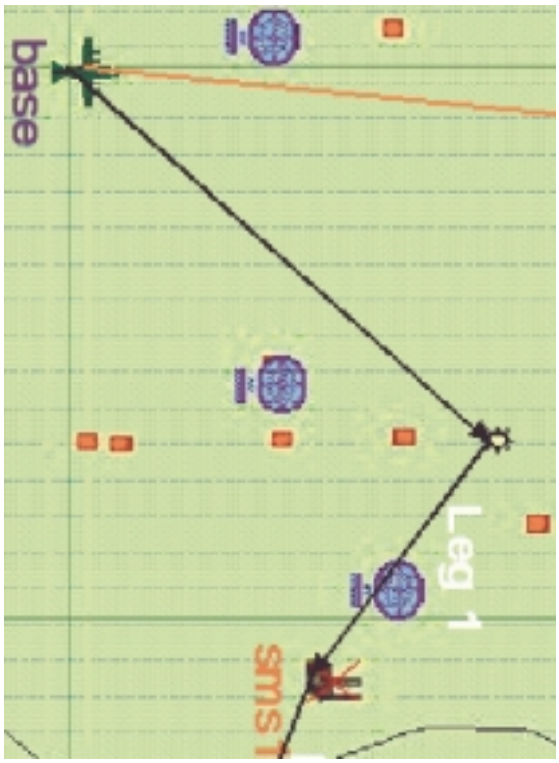


Figure 20: Leg example.

```
type task_simulation
{
 output
   ucav u1, u2, u3, u4;
   leg leg1, leg2, leg3, leg4, leg5, leg6, leg7, leg8;
   subtask subtask1, subtask2;
   task_controller c_task1;
   task t1;
   set(ucav) team1:={}, team2:={};

 discrete
   i1, i2, i3, i4, normal;

 transition
 i1 -> i2 {} do            // create all legs
  {
    leg1:= create(leg, p_attack:= [ [93517.725, 111320.00],
                                    [150000.00, 158235.2],
                                    [151000.00, 158000.00]],
                                     p:=medium_sam12);
     leg2:= create(leg, p_attack:= [], p:=long_sam5_trk);
    .....
    team1:={u1, u2};  // create teams to execute subtasks
     team2:={u3, u4};
    },

 i2 -> i3 {} do // creates subtasks and leg dependencies
  {
   subtask1:= create(subtask,
                        p:=[leg1, leg2, leg3, leg4, leg5],
                        team:= team1);
   requires(leg3):={leg6};
   subtask2:= create(subtask,
                       p:=[leg6, leg7, leg8], team:= team2);
  },
 i3 -> i4 {} do                  // creates task specification
  {
   t1:=create(task,s:=[subtask1,subtask2]);
  },
  i4 -> normal {} do             // creates task controller
    {
     c_task1:= create(task_controller, t:=t1);
    };
}
```

Figure 21: An example task specification: the task comprises subtasks 1 and 2 to which are assigned teams 1 and 2; each task consists of several legs.

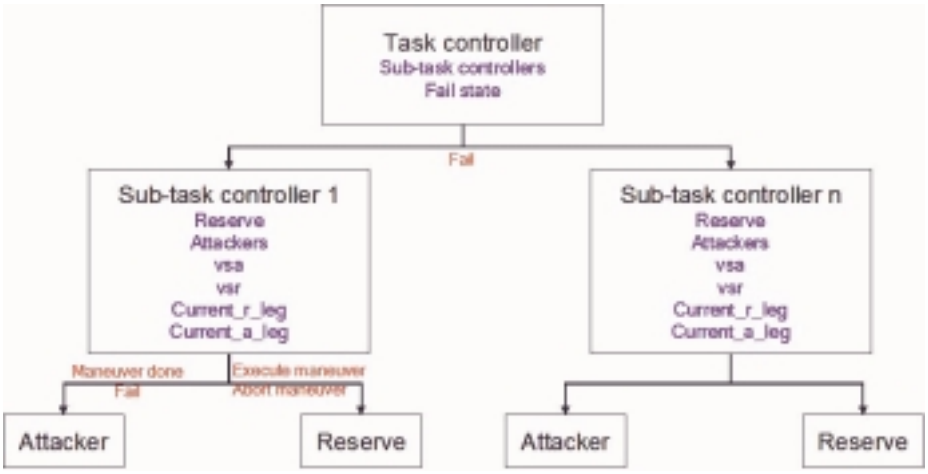


Figure 22: Task controller.

6 Module 4: State estimator

From the viewpoint of the Blue forces, the 'state-of-the-world' has two components during task execution: the state of the Blue forces themselves—which we assume is fully known, and the Red forces. The latter is only partially known and, since we adopt a Bayesian viewpoint, this knowledge is fully described by the threat probability distribution. This distribution changes as tasks are executed.

In this section we describe a procedure, called $Post_{threat}$, that updates the threat probability distribution following the destruction of some targets or the acquisition of sensor measurement.

Recall our assumption about the initial threat distribution in section 3.1, which is repeated here:

> The Red force is distributed over areas $A_1, \cdots, A_k$. In area A_j there are N_{tj} targets of type $t \in$ *TargetTypes* whose locations are independently and uniformly distributed. The random number of targets N_{tj} are all independent with distribution $P_{tj}(N)$.

This restriction implies that the initial threat distribution has the form

$$P_{threat}(0)(Targets) = \prod_{t} \prod_{j=1}^{k} \prod_{i=1}^{N_{tj}} p_{tj}(type = t, (x_i, y_i)) P_{tj}(N_{tj}), \tag{32}$$

in which t ranges over *TargetTypes*, and

$$p_{tj}(type = t, (x_i, y_i)) = \begin{cases} |A_j|^{-1}, & (x_i, y_i) \in A_j \\ 0, & \text{otherwise} \end{cases} . \tag{33}$$

The procedure $Post_{threat}$ takes two arguments—the initial threat distribution $P_{threat}(0)$ and a task list—and calculates the resulting distribution. We will need to model what it means to execute a task.

We consider two kinds of tasks: *strike* and *search*. The former task results in the destruction of some targets with success probability S_{min}; the latter results in sensor observations that reduce the uncertainty in the distribution. As we will see, a useful feature of the model (32)-(33) is that the threat distribution following the execution of a task also has the same form.

Denote the initial distribution by

$$P_{threat}(0) = P_{A,N}. \tag{34}$$

6.1 Threat distribution after strike

There are two cases to consider.

$S_{\textbf{\textit{min}}} = 1$

When the success probability is very high, $S_{min} \approx 1$, the procedure is straightforward. The strike task is completed by the destruction of a certain number of targets of each type in each area. So this task is specified by a set of the form

$$\textit{strike} = \{(t, A_j, n_{tj}) \mid t \in \textit{TargetTypes}; j = 1, \cdots, k\}. \tag{35}$$

The successful completion of task (35) reduces the number of targets, so the posterior threat will have the same form as (34), except that the random variables N_{tj} will be reduced:

$$\textit{Post}_{\textit{threat}}(P_{A,N}, \textit{strike}) = P_{A,(N-n)^+}, \tag{36}$$

in which $(N-n)^+ = \{\max\{N_{tj} - n_{tj}, 0\}\}$.

If *TaskList* contains m *strike* tasks,

$$\textit{TaskList} = \{\textit{strike}_1, \cdots, \textit{strike}_m\},$$

the posterior threat is evaluated by successively applying formula (36), so

$$\textit{Post}_{\textit{Threat}}(P_{A,N}, \textit{TaskList}) = P_{A,(N-n_1-\cdots-n_k)^+}.$$

The condition $S_{min} \approx 1$ is achieved either if there is a very high confidence in the success of a strike, or if a strike is followed by an accurate damage assessment, which removes any uncertainty in the success of a strike.

$S_{\textbf{\textit{min}}} < 1$

The task $\textit{strike} = \{(\textit{SAM}, A, 1)\}$ calls for the destruction of one SAM site in area A. If initially there are N SAM sites in A and the task is successful, which happens with probability S_{min}, it will reduce this number to $(N-1)^+$. If the task is unsuccessful, this number will remain N. So the posterior threat is

$$\textit{Post}_{\textit{threat}}(P_{A,N}, \textit{strike}) = P_{A,\textit{Post}(N)},$$

in which $\textit{Post}(N)$ is the random number[14]

$$\textit{Post}(N) = (N-1)^+ 1(\textit{success}) + N[1 - 1(\textit{success})],$$

in which *success* is a $\{0,1\}$-valued random variable, independent of N, with $P(\textit{success} = 1) = S_{min}$.

More generally, if the strike task is given by (35), the posterior threat evaluates to

$$\textit{Post}_{\textit{threat}}(P_{A,N}, \textit{strike}) = P_{A,\textit{Post}(N)}, \tag{37}$$

in which

$$\textit{Post}(N) = (N-n)^+ 1(\textit{success}) + N[1 - 1(\textit{success})]. \tag{38}$$

[14] In the formula below, $1(success) = 1$ if $success = 1$, and equals 0, otherwise.

If *TaskList* contains several strike tasks, the posterior threat is evaluated by successively applying formulas (37)-(38).[15]

Equations (35)-(38) summarize the procedure $\mathit{Post}_{threat}(P_{A,N}, \mathit{TaskList})$ for the situations in which *TaskList* consists of strike tasks.

6.2 Threat distribution after search

We calculate $\mathit{Post}_{\mathit{threat}}(P_{A,N}, \mathit{search})$ after the completion of a search task.

To simplify the notation we assume that the search is confined to one area in the list $A = \{A_1, \cdots, A_k\}$. Because the threats in different areas are independent, the threat distribution will remain the same in all the areas that are not searched. So we may assume that the list consists of a single area, also denoted by A.

Suppose now that the area A is divided into two disjoint subareas A_1 and A_2, and the search is confined to area A_2. We assume that the search is able to distinguish between targets of different types. So the result of the search is a list of 'observations' $Y = \{Y_t \mid t \in \mathit{TargetTypes}\}$, where Y_t is the number of observed targets of type t. Because the targets of different types in an area are independent, we may calculate the effect of each observation Y_t separately.

Thus suppose there is only a single type of target and the result of the search is the number of targets Y observed in area A_2. The observation need not be perfect, i.e., not all targets in A_2 may be observed. Rather, we assume that the probability of detecting a target is p_d. Therefore if N_2 is the (random) number of targets in A_2, Y is related to N_2 by

$$P(Y = y \mid N_2 = n_2) = \begin{cases} \binom{n_2}{y}(p_d)^y(1-p_d)^{n_2-y}, & y \le n_2 \\ 0, & y > n_2. \end{cases} \tag{39}$$

Equation (39) is our model of the search process. Note that it may serve equally as a model of a sensor observation.

Our prior information is simply $P_{A,N}$ which, becaue we are considering a single area and a single target type, simply reduces to the probability distribution $P(N = n)$ of the number of targets in A. Let N_i be the (random) number of targets in area A_i, $i = 1, 2$.

The prior marginal distribution of N_i is

$$P(N_i = n_i) = \sum_{n=0}^{\infty} P(N_i = n_i \mid N = n)P(N = n), \tag{40}$$

in which

$$P(N_i = n_i \mid N = n) \begin{cases} \binom{n}{n_i}(\alpha_i)^{n_i}(1-\alpha_i)^{n-n_i}, & n_i \le n \\ 0, & n_i > n, \end{cases} \tag{41}$$

in which $\alpha_i = |A_i|/|A|$ is the fraction of the area A that is in A_i.

After $Y = y$ targets are observed in A_2, the threat distribution in area A_i is given by $P(N_i = n_i \mid Y = y)$, which we want to calculate.

[15] Note that if the planner believes that the success of the different tasks in *TaskList* are correlated, the distribution of *Post(N)* must take this correlation into account.

Example 5. The calculation is easy in the case of perfect detection, $p_d = 1$. We have

$$\begin{aligned} P(N_2 = y \mid Y = y) &= 1, \text{ and} \\ P(N_1 = n_1 \mid Y = y) &= P(N = n_1 + y \mid N \geq y) \\ &= \frac{P(N = n_1 + y)}{\sum_{n=y}^{\infty} P(N = n)} \end{aligned}$$

In the general case, $p_d < 1$, the posterior distribution of N_2 is given by Bayes' rule:

$$\begin{aligned} P(N_2 = n_2 \mid Y = y) &= \frac{P(Y = y \mid N_2 = n_2)P(N_2 = n_2)}{P(Y = y)} \\ &= \frac{P(Y = y \mid N_2 = n_2)P(N_2 = n_2)}{\sum_{m=0}^{\infty} P(Y = y \mid N_2 = m)P(N_2 = m)}. \end{aligned} \tag{42}$$

The terms on the right hand side of (42) are given by (39), (40), 41).

The calculation of $P(N_1 = n_1 \mid Y = y)$ is a bit more complicated. We have

$$\begin{aligned} P(N_1 = n_1 \mid Y = y) &= \sum_{n_2=0}^{\infty} P(N_1 = n_1 \mid N_2 = n_2, Y = y)P(N_2 = n_2 \mid Y = y) \\ &= \sum_{n_2=0}^{\infty} P(N_1 = n_1 \mid N_2 = n_2)P(N_2 = n_2 \mid Y = y), \end{aligned} \tag{43}$$

because N_1 and Y are conditionally independent give N_2. Also,

$$P(N_1 = n_1 \mid N_2 = n_2) = \frac{P(N_1 = n_1, N_2 = n_2)}{P(N_1 = n_2)}, \tag{44}$$

and

$$\begin{aligned} P(N_1 = n_1, N_2 = n_2) &= \sum_{n=0}^{\infty} P(N_1 = n_1, N_2 = n_2 \mid N = n)P(N = n) \\ P(N_1 = n_1, N_2 = n_2 \mid N = n) &= \begin{cases} \frac{\alpha_1^{n_1}\alpha_2^{n_2}}{\sum_{n_1+n_2=n}\alpha_1^{n_1}\alpha_2^{n_2}}, & \text{if } n_1 + n_2 = n \\ 0, & \text{if } n_1 + n_2 \neq n \end{cases} \end{aligned} \tag{45}$$

The distribution $P(N_1 = n_1 \mid Y = y)$ is obtained by substituting from (41), (42), (44), (45) into (43).

6.3 Implementation

The procedure $\mathit{Post}_{\mathit{threat}}(P_{A,N}, \mathit{task})$ is implemented in a database, as described in section 2.4. The database stores the threat distribution, and updates it following acquisition of information. Crucial to the reduction in memory requirements and computational complexity, is the representation (32). If there are α areas, τ target types, and at most ν targets in each area of each type, the distribution (32) can be stored in an array of size $\alpha \times \tau \times \nu$.

7 Module 5: Java interface to OEP

A full planning cycle involves these steps:

1. The planner creates a set of paths for attack using the ITP;
2. The set of paths and the threat elimination matrix are used by the task assignment module to create a set of assignments of teams to tasks and a partial ordering of the tasks;
3. The Shift controller set uses the information from the assignment module to execute the plan with the help of the java RMI services and the java client to the OEP;
4. Feedback data from the OEP is placed in the database, and when the Shift controller concludes execution, this data is read into the ITP to run another planning iteration using this updated information.

This section provides an explanation of the functionality of the java RMI services and the java client to the OEP.

7.1 Java client to the OEP

The java client functions as the 'glue' between the OEP and the Shift controllers, and also provides the mechanism for updated sensor readings from the OEP to be stored in the database using the RMI services. During initialization of the Shift controllers, one instance of the java client is created for each platform in the scenario. This is accomplished using the Java Native Interface from the Shift runtime. Each instance of the java client is initialized with the current state of the platform from the OEP. When the java client is initialized, each client object connects to the database (through JDBC drivers for MySQL) and to the OEP using the CORBA naming service. It also obtains references to the RMI services using the naming service. Initial values for variables in each client object are set in the Shift controller, and then updated from the OEP using the OEP events mechanism. Each client object has a get() and set() method, which is used by the Shift runtime to maintain a mirror of the OEP state within Shift. In the get() method of each client are included calls to the RMI services to update the database tables.

In the scenario being run in the OEP, we initialize a wide_body_isr that flies in a holding pattern while reading sensor data into a queue. Each client object subscribes to events for the platform it corresponds to and for the wide_body_isr platform. When the wide body platform receives a sensor reading, the sensor data is added to a queue in the client object. When the queue reaches a predetermined size, the java client invokes the threat distribution RMI service to read the updated values from the sensor readings into the database. When the queue is empty of new values, the risk map update service is called to recalculate the risk map. Barring interruptions for database update, execution continues for a time interval specified in the client object before new values from the OEP are read into the Shift runtime.

7.2 RMI Services

The java RMI services provide two basic functions: one service is a database interface that enables the java client to the OEP to add and remove threat information from the threat distribution table. The other service recalculates

the values in each cell of the risk map by reading updated threat information from the database table and writes the updated values in the risk map back to the database. The RMI services have to be started before starting execution of the Shift controllers. Appropriate user interfaces are provided for the user to start the RMI services while specifying the scenario area, grid size and other necessary parameters. The database tables are initialized according to the values provided for each of the parameters. Typically, the scenario area is divided into a grid (101×101) and the database contains a threat distribution table and a risk map table that contain a value for each cell in the grid.

8 Module 6: Robust path planning

Modules 1-5, described the previous sections provide an integrated but incomplete approach to the MICA problem. Modules 6-8, discussed in this and the two subsequent sections, represent theoretical formulations and preliminary algorithms that we had planned to integrate with the other modules.

Module 6 considers a formulation of Markov Decision Problems, when there is uncertainty about the various transition probabilities that model the underlying process. Module 7 is concerned with 'flexible' team formation, based on information about force attritions that is obtained as task execution proceeds. Module 8 presents a path planning algorithm that deals with multiple criteria.

Optimal solutions to Markov Decision Problems (MDPs) may be very sensitive with respect to the state transition probabilities. In many practical problems, the estimation of these probabilities is far from accurate. Hence, estimation errors are limiting factors in applying MDPs to real-world problems. We consider the problem of minimizing the worst-case (maximum) expected cost, where the maximum is taken over all possible time-independent choices of the transition matrices within prescribed convex sets. We derive a robust counterpart to the classical Bellman recursion, based on approximating the original problem by one where the uncertain transition matrices are allowed to be time-dependent. Our main result is that a particular choice of the convex sets used to represent uncertainty, based on likelihood or entropy bounds, leads to both a statistically meaningful representation of uncertainty, and a complexity of the robust recursion that is similar to that of the classical recursion. Hence, robustness can be added at practically no extra computing cost.[16]

8.1 Introduction

Finite-state and finite-action Markov Decision Processes (MDPs) capture several attractive features that are important in decision-making under uncertainty: they handle risk in sequential decision-making via a state transition probability matrix, while taking into account the possibility of information gathering and recourse corresponding to this information during the multi-stage decision process [29, 31, 32, 39].

Module 6 addresses the issue of uncertainty at a higher level: we consider a Markov decision problem in which the transition matrix itself is uncertain, and seek a robust decision for it. Our work is motivated by the fact that in most practical problems, the transition matrix has to be estimated from data, which is often a difficult task, see for example [49, 39, 56, 58]. It turns out that estimation errors may have a huge impact on the solution, which is often quite sensitive to changes in the transition probabilities. (We will provide an example of this phenomenon.)

A number of authors have addressed the issue of uncertainty in the transition matrix. A Bayesian approach such as described by [51] requires a perfect knowledge of the whole prior distribution on the transition matrix, making it difficult to apply in practice. Other authors have considered the transition matrix to lie in a given set, most typically a polytope: see [57, 53, 60]. Although our approach allows one to describe the uncertainty on the transition matrix by a polytope, we will argue *against* choosing such a model for the uncertainty. First, a polytope is often not a tractable way to address the robustness problem, as it incurs a significant additional computational effort to handle

[16]Research on module 6 was conducted by L. El Ghaoui and A. Nilim and was also supported in part by Eurocontrol-014692 and NSF-ECS-9983874.

uncertainty. As we will show, an exception is when the uncertainty is described by an interval matrix, intersected by the constraint that probabilities sum to one, as in [60, 61]. Perhaps more importantly, polytopic models, especially interval matrices, are very poor representations of statistical uncertainty and lead to very conservative robust policies. In addition, in [61] the authors proposed relative entropy models and proved the existence of a polynomial time solution. However, no specific algorithms were proposed.

We propose here an uncertainty model which results in an algorithm that is *both* statistically accurate and numerically tractable. We develop a formulation in which the concern for robustness can be handled at virtually no additional computational cost. This means that the method is directly applicable to those problems already amenable to exact dynamic programming via Bellman recursions.

This section is organized as follows. The problem is set up in section 8.2. In sections 8.4, we describe the so-called likelihood model and some variations. Section 8.6 examines the entropy models, while section 8.7 deals with ellipsoidal and "interval matrix" models. Our results are summarized in section 8.3. We describe numerical results in the context of aircraft routing in section 8.10.

Notation

$P > 0$ or $P \geq 0$ refers to the strict or non-strict componentwise inequality for matrices or vectors. For a vector $p > 0$, $\log p$ refers to the componentwise operation.

8.2 Problem Setup

8.2.1 The Bellman recursion

We consider a finite horizon Markov decision process with finite state and finite action sets. The decision horizon is divided into a finite number of stages $T = \{0, 1, 2, \ldots, N\}$, for some $1 \leq N < \infty$. At each stage, the system occupies a state $i \in \mathcal{X}$; $n = |\mathcal{X}|$ is finite. At each stage and state, a decision maker is allowed to choose an action a deterministically from a finite set of allowable actions in state i, $\mathcal{A}_i$. Let $\mathcal{A} = \cup_i \mathcal{A}_i$ and let $m = |\mathcal{A}|$. We denote by $P = (P^a)_{a \in \mathcal{A}}$ the collection of transition matrices, by $\pi_t = (a_t, a_{t+1}, \ldots, a_{N-1})$ the policy starting from time $t \in T$, by Π_t the strategy space at time $t \in T$, by $c_t(i, a)$ the cost corresponding to state $i \in \mathcal{X}$ and action $a \in \mathcal{A}$ at time $t \in T$, and by c_N the cost function at the terminal stage, N. We assume that the cost function is finite everywhere.

For a given state $i \in \mathcal{X}$ and action $a \in \mathcal{A}$, we denote by p_i^a the next-state distribution drawn from P^a corresponding to state $i \in \mathcal{X}$; thus p_i^a is the i-th row of matrix P^a. We denote by $\mathcal{P}_i^a$ the projection of the set $\mathcal{P}^a$ onto the set of p_i^a-variables.

Our *nominal* problem is to minimize the expected cost over a finite horizon

$$\min_{\pi \in \Pi} \mathbf{E} \left(\sum_{t=0}^{N-1} c_t(i_t, a_t) + c_N(i_N) \right),$$

in which $\pi := \pi_0$, and $\Pi_0 := \Pi$. When the transition matrices are exactly known, the value function of the system at state $i \in \mathcal{X}$ and the stage $t \in T$ can be computed via the Bellman recursion,

$$V_t(i) = \min_{a \in \mathcal{A}} \left(c_t(i,a) + \sum_{j=1}^{n} P^a(i,j) V_{t+1}(j) \right), \quad i \in \mathcal{X}. \tag{46}$$

Each step of the Bellman recursion has worst-case complexity $O(nm)$.

8.2.2 Addressing uncertainty in the transition matrices

Now consider the case when for each action a, the corresponding transition matrix P^a is only known to lie in some given convex and compact subset $\mathcal{P}^a$ of $\mathcal{T}$, where $\mathcal{T}$ is the set of $n \times n$ transition matrices (componentwise non-negative matrices with rows summing to one). Loosely speaking, we can think of the sets $\mathcal{P}^a$ as *sets of confidence* for the transition matrices.

Two models for transition matrix uncertainty are possible, leading to two possible forms of robust control problems. In a first model, referred to as the *time-invariant* model, the transition matrix for any given action is chosen by nature once and for all, and remain fixed thereafter. This means that the transition matrix depends only on the action taken by the controller. In a second model, which we refer to as the *time-varying* model, the transition matrices can vary arbitrarily with time, within their prescribed bounds. In that case, the transition matrix depends not only on the controller's action, but also on time.

Let us define our two problems more formally. For a given policy $\pi \in \Pi$, we define *nature's policy* as $\mathbf{P}(\pi) = (P_0^{a_0}, \ldots, P_{N-1}^{a_{N-1}})$, which corresponds to the transition matrices chosen by nature in response to a given control policy π. Define the corresponding set of allowable policies by $\mathcal{P}(\pi) = \mathcal{P}^{a_0} \otimes \ldots \otimes \mathcal{P}^{a_{N-1}}$. Finally, define the set of time-invariant policies by

$$\mathcal{L}(\pi) = \left\{ (P_0^{a_0}, \ldots, P_{N-1}^{a_{N-1}}) \in \mathcal{T}^N \ : \ P_i^{a_i} = P_j^{a_j} \text{ for } a_i = a_j \right\}.$$

The time-invariant model leads to the problem

$$\min_{\pi \in \Pi} \max_{\mathbf{P}(\pi) \in \mathcal{P}(\pi) \bigcap \mathcal{L}(\pi)} \mathbf{E} \left(\sum_{t=0}^{N-1} c_t(i_t, a_t) + c_N(i_N) \right). \tag{47}$$

In contrast, the time-varying model leads to a relaxed version of the above:

$$\min_{\pi \in \Pi} \max_{\mathbf{P}(\pi) \in \mathcal{P}(\pi)} \mathbf{E} \left(\sum_{t=0}^{N-1} c_t(i_t, a_t) + c_N(i_N) \right). \tag{48}$$

The first model is attractive for statistical reasons, as it is much easier to develop statistically accurate sets of confidence when the underlying process is time-invariant. Unfortunately, the resulting game (47) seems to be hard to solve. The second model is attractive as one can solve the corresponding game (48) using a variant of Bellman

recursion seen below, but we are left with a difficult task, that of estimating a meaningful set of confidence for the time-varying matrices P^a.

In this section, we use the first model of uncertainty, in which the transition matrix is fixed. This allows us to describe uncertainty in a statistically accurate way using likelihood or entropy functions. To solve the corresponding control problem (47), we use an approximation that is common in robust control, wherein the time-invariant uncertainty is replaced by a time-varying one. This means that we solve the second problem (48) as an approximation (upper bound) to the first, using uncertainty sets $\mathcal{P}^a$ derived from a time-invariance assumption about the transition matrices.

8.2.3 The robust Bellman recursion

Problem (48) is a two-player stochastic game with non-negative, finite rewards. By standard arguments from stochastic game theory [30], this game can be viewed as a zero sum game. General results from [52, 54] then imply that the corresponding stochastic game can be solved via the following "robust counterpart" to the Bellman recursion,

$$V_t(i) = \min_{a \in \mathcal{A}} \max_{p \in \mathcal{P}_i^a} \left(c_t(i,a) + \sum_{j=1}^{n} p(j) V_{t+1}(j) \right), \quad i \in \mathcal{X}, \tag{49}$$

in which $V_t(i)$ is the worst-case optimal value function in state i at stage t. The above result is proved in Appendix 8.16.1.

One step of the robust Bellman recursion thus involves the solution of a convex optimization problem. Obviously, the complexity of the robust Bellman recursion depends solely on the complexity of the projections $\mathcal{P}_i^a$ for each $i \in \mathcal{X}$ and $a \in \mathcal{A}$. Moreover, the set $\mathcal{P}$ should be an accurate (non-conservative) description of the statistical uncertainty on the whole collection of transition matrices.

Note that the effect of uncertainty on a *given* strategy $\pi_t = (a_t, \ldots, a_N)$ can be evaluated by the following recursion

$$V_t^{\pi_t}(i) \quad = \quad \max_{p \in \mathcal{P}_i^{a_t}} \left(c_t(i,a_t) + \sum_{j=1}^{n} p(j) V_{t+1}^{\pi_{t+1}}(j) \right), \quad i \in \mathcal{X}, \tag{50}$$

which provides the worst-case value function for a given strategy.

8.2.4 Main result

We address the problem of efficiently computing the value function via the above recursion. Once the uncertainty model is chosen, the challenge is to solve the "inner problem" in (49), which reduces to computing values of the support function of a given convex set $\mathcal{U}$:

$$\phi_{\mathcal{U}}(v) = \max_{p \in \mathcal{U}} v^T p, \tag{51}$$

in which the variable p corresponds to a particular row of a specific transition matrix, $\mathcal{U}$ is the set that describes the uncertainty on this row, and v is an appropriately defined vector, containing the elements of the value function.

We consider various representations of uncertainty. All our models involve *independent* descriptions of the uncertainty on each transition matrix; in other words, we postulate that $\mathcal{P}$ is a direct product $\bigotimes_{a\in\mathcal{A}} \mathcal{P}^a$, in which $\mathcal{P}^a$ describes uncertainty on the transition matrix P^a. This assumption is not formally needed, but simplifies the task of forming the projections $\mathcal{P}_i^a$ required in the robust Bellman recursion (49).

Our main uncertainty model is based on a log-likelihood constraint on each transition matrix. This representation enables one to solve for one step the robust dynamic programming recursion (49) in worst-case time of $O(n\log(1/\epsilon))$ via a simple bisection algorithm, where n is the size of the state space, and ϵ specifies the accuracy of the worst-case value function. This brings the total complexity of one step of the Bellman recursion to $O(nm\log(1/\epsilon))$, where m is the cardinality of the action set. At the same time, our model allows an accurate description of statistical uncertainty on the transition matrix. Hence, non-conservative robustness is obtained at a moderate increase ($\log(1/\epsilon)$) with respect to the classical Bellman recursion. We also describe models based on relative entropy bounds, and obtain similar results.

We will also consider perhaps more classical ways to describe uncertainty, among which an interval models based on componentwise intervals of confidence, and ellipsoidal models that are based on quadratic approximations to the log-likelihood. We will observe that some of these descriptions give rise to similar low complexity results. However, these "approximate" models, as argued below, are statistically less accurate.

8.3 Robust algorithm summary

The robust Dynamic Programming algorithm is as follows.

1. Initialize the value function v_t to its terminal value v_N.
2. Repeat until $t = 0$:
   (a) For all states i and controls a, compute the solution to the inner problem
   $$\phi(v_t) = \max_{p\in\mathcal{P}_i^a} p^T v_t;$$
   (b) Update the value function by
   $$v_{t-1}(i) = \min_{a\in\mathcal{A}} \left(c_{t-1}(i,a) + \phi(v_t)\right);$$
   (c) Replace t by $t-1$ and go to 2.

8.4 Likelihood Models

Our first model is based on a likelihood constraint to describe uncertainty on each transition matrix. Our uncertainty model is derived from a controlled experiment starting from state $i = 1, 2, \ldots, n$ and the count of the number of transitions to different states. We denote by F^a the matrix of empirical frequencies of transition with control a in

the experiment; denote by f_i^a its i^{th} row. We have $F^a \geq 0$ and $F^a\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ denotes the vector of ones. For simplicity, we assume that $F^a > 0$ for every a.

To simplify the notation, we will drop the superscript a in this section, and refer to a generic transition matrix as P and to its i^{th} row as p_i. The same convention applies to the empirical frequency matrix F^a and its rows f_i^a, as well as to sets $\mathcal{P}^a$ and $\mathcal{P}_i^a$. When the meaning is clear from context, we will further drop the subscript i.

8.4.1 Model description

The "plug-in" estimate $\hat{P} = F$ is the solution to the maximum likelihood problem

$$\max_P \; L(P) := \sum_{i,j} F(i,j) \log P(i,j) \;:\; P \geq 0, \quad P\mathbf{1} = \mathbf{1}$$

The optimal log-likelihood is $\beta_{\max} = \sum_{i,j} F(i,j) \log F(i,j)$.

A classical description of uncertainty in a maximum-likelihood setting is via the likelihood region [35, 38]

$$\left\{ P \in \mathbf{R}^{n\times n} \;:\; P \geq 0, \quad P\mathbf{1} = \mathbf{1}, \quad \sum_{i,j} F(i,j) \log P(i,j) \geq \beta \right\}, \tag{52}$$

in which $\beta < \beta_{\max}$ is a chosen number, which represents the designers preferred uncertainty level. In practice, the designer chose an uncertainty level and β can be estimated using resampling methods, or a large-sample Gaussian approximation, so as to ensure that the set above achieves the desired level of confidence (see Appendix 8.16.4).

The description above is classical in the sense that log likelihood regions are the starting point for developing ellipsoidal or interval models of confidence, hence are statistically more accurate [35]; see section 8.9 for further details. The set (52) tells us how informative the data is. If this set is elongated along a direction, then the likelihood function does not vary much in that direction, and the the data is not very informative in that direction. This set has some interesting features. First, it does not result from a (quadratic) approximation; it is a valid description of uncertainty, even for β values that are far below $\beta_{\max}$. Second, this set might not be symmetric around the maximum-likelihood point, reflecting the fact the statistical uncertainty depends on the direction. Finally, by construction, it excludes matrices that are not transition matrices; the same cannot be said of the more classical ellipsoidal approximations.

In our problem, we only need to work with the uncertainty on each row p_i, that is, with *projections* of the set above. Due to the separable nature of the maximum-likelihood problem, the projection of the set (52 onto the p_i variables of matrix P can be given explicitly as

$$\mathcal{P}_i(\beta_i) := \left\{ p \in \mathbf{R}^n \;:\; p \geq 0, \quad p^T\mathbf{1} = 1, \quad \sum_j f_i(j) \log p_i(j) \geq \beta_i \right\},$$

in which

$$\beta_i := \beta + \sum_{k\neq i} \sum_j F(k,j) \log F(k,j).$$

8.4.2 The dual problem

We are now ready to attack problem (51) under the premise that the transition matrix is only known to lie in some likelihood region as defined above. The inner problem is to compute

$$\phi := \max_p \; p^T v \;:\; p \geq 0, \;\; p^T \mathbf{1} = 1, \;\; \sum_j f(j) \log p(j) \geq \beta,$$

in which we have dropped the subscript i in the empirical frequencies vector f_i and in the lower bound β_i. In this section $\beta_{\max}$ denotes the maximal value of the likelihood function appearing in the set above, which is $\beta_{\max} = \sum_j f(j) \log f(j)$. We assume that $\beta < \beta_{\max}$, which, together with $f > 0$, ensures that this set has non-empty interior.

The Lagrangian $\mathcal{L} : \mathbf{R}^n \times \mathbf{R}^n \times \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ associated with the inner problem can be written as

$$\mathcal{L}(v, \nu, \mu, \lambda) = p^T v + \nu^T p + \mu(1 - p^T \mathbf{1}) + \lambda(f^T \log p - \beta),$$

in which ν, μ, and λ are the Lagrange multipliers. The Lagrange dual function $d : \mathbf{R}^n \times \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ is the maximum value of the Lagrangian over p, i.e., for $\nu \in \mathbf{R}^n$, $\mu \in \mathbf{R}$, and $\lambda \in \mathbf{R}$,

$$d(\nu, \mu, \lambda) = \sup_p \mathcal{L}(v, \nu, \mu, \lambda) = \sup_p (p^T v + \nu^T p + \mu(1 - p^T \mathbf{1}) + \lambda(f^T \log p - \beta)). \tag{53}$$

The optimal $p^* = \arg\sup_p \mathcal{L}(v, \nu, \mu, \lambda)$ is readily obtained by solving $\frac{\partial \mathcal{L}}{\partial p} = 0$, which results in

$$p^*(i) = \frac{\lambda f(i)}{\mu - v(i) - \nu(i)}.$$

Plugging the value of p^* in the equation for $d(\nu, \mu, \lambda)$ yields, with some simplification, the following dual problem:

$$\overline{\phi} := \min_{\lambda, \mu, \nu} \; \mu - (1 + \beta)\lambda + \lambda \sum_j f(j) \log \frac{\lambda f(j)}{\mu - v(j) - \nu(j)} \;:\; \lambda \geq 0, \;\; \nu \geq 0, \;\; \nu + v \leq \mu \mathbf{1}.$$

Since this problem is convex, and has a feasible set with non-empty interior, there is no duality gap, that is, $\phi = \overline{\phi}$. Moreover, by a monotonicity argument, the optimal dual variable ν is zero, which reduces the number of variables to two:

$$\phi = \min_{\lambda, \mu} \; h(\lambda, \mu),$$

in which

$$h(\lambda, \mu) := \begin{cases} \mu - (1 + \beta)\lambda + \\ \lambda \sum_j f(j) \log \dfrac{\lambda f(j)}{\mu - v(j)} & \text{if } \lambda > 0, \;\; \mu > v_{\max} := \max_j v(j), \\ +\infty & \text{otherwise.} \end{cases} \tag{54}$$

For future reference, we note that h is twice differentiable on its domain, and that its gradient is given by

$$\nabla h(\lambda, \mu) = \begin{bmatrix} \sum_j f(j) \log \dfrac{\lambda f(j)}{\mu - v(j)} - \beta \\ 1 - \lambda \sum_j \dfrac{f(j)}{\mu - v(j)} \end{bmatrix}. \tag{55}$$

8.4.3 A bisection algorithm

From the expression of the gradient obtained above, the optimal value of λ for a fixed μ, $\lambda(\mu)$, is given analytically by

$$\lambda(\mu) = \left(\sum_j \frac{f(j)}{\mu - v(j)} \right)^{-1}, \tag{56}$$

which further reduces the problem to a one-dimensional problem,

$$\phi = \min_{\mu \geq v_{\max}} \phi(\mu),$$

in which $v_{\max} = \max_j v(j)$, and $\phi(\mu) = h(\lambda(\mu), \mu)$. By construction, the new function $\phi(\mu)$ is convex in its (scalar) argument, since the function h defined in (54) is jointly convex in both its arguments (see [40, p.74]). Hence, we may use bisection to minimize ϕ.

To initialize the bisection algorithm, we need upper and lower bounds μ_- and μ_+ on a minimizer of ϕ. When $\mu \to v_{\max}$, $\phi(\mu) \to v_{\max}$ and $\phi'(\mu) \to -\infty$ (see Appendix 8.16.2). Thus, we may set the lower bound to $\mu_- = v_{\max}$.

The upper bound μ_+ must be chosen such that $\phi'(\mu_+) > 0$. We have

$$\phi'(\mu) = \frac{\partial h}{\partial \mu}(\lambda(\mu), \mu) + \frac{\partial h}{\partial \lambda}(\lambda(\mu), \mu) \frac{d\lambda(\mu)}{d\mu}.$$

The second term is zero by construction, and $d\lambda(\mu)/d\mu > 0$ for $\mu > v_{\max}$. Hence, we only need a value of μ for which

$$\frac{\partial h}{\partial \lambda}(\lambda(\mu), \mu) = \sum_j f(j) \log \frac{\lambda(\mu) f(j)}{\mu - v(j)} - \beta > 0. \tag{57}$$

By convexity of the negative log function, and using the fact that $f^T \mathbf{1} = 1$, $f \geq 0$, we obtain

$$\begin{aligned} \frac{\partial h}{\partial \lambda}(\lambda(\mu), \mu) &= \beta_{\max} - \beta + \textstyle\sum_j f(j) \log \frac{\lambda(\mu)}{\mu - v(j)} \\ &\geq \beta_{\max} - \beta - \log \left(\textstyle\sum_j f(j) \frac{\mu - v(j)}{\lambda(\mu)} \right) \\ &\geq \beta_{\max} - \beta + \log \frac{\lambda(\mu)}{\mu - \bar{v}}, \end{aligned}$$

in which $\bar{v} = f^T v$ denotes the average of v under f.

The above, combined with the bound on $\lambda(\mu)$: $\lambda(\mu) \geq \mu - v_{\max}$, yields a sufficient condition for (57) to hold:

$$\mu > \mu_+^0 := \frac{v_{\max} - e^{\beta - \beta_{\max}} \bar{v}}{1 - e^{\beta - \beta_{\max}}}. \tag{58}$$

By construction, the interval $[v_{\max}\ \mu_+]$ is guaranteed to contain a global minimizer of ϕ over $(v_{\max}\ +\infty)$.

The bisection algorithm goes as follows:

1. Set $\mu_- = v_{\max}$ and $\mu_+ = \mu_+^0$ as in (58). Let $\epsilon > 0$ be a small convergence parameter.
2. While $\mu_+ - \mu_- > \epsilon(1 + \mu_- + \mu_-)$, repeat
   (a) Set $\mu = (\mu_+ + \mu_-)/2$.
   (b) Compute the gradient of ϕ at μ.
   (c) If $\phi'(\mu) > 0$, set $\mu_+ = \mu$; otherwise, set $\mu_- = \mu$.
   (d) go to 2a.

Each iteration of the above algorithm has worst-case complexity of $O(n)$. The number of iterations grows as $\log(\mu_+^0 - v_{\max})/\epsilon)$, which is independent of problem size. Hence, the worst-case complexity of the algorithm is $O(n)$, which is the same order as one evaluation of the objective function. Therefore, the extra cost of adding robustness under the likelihood uncertainty model is $O(1)$, which means that robustness can be added at practically no extra cost.

In practice, the function to minimize may be very "flat" near the minimum. This means that the bisection algorithm above may take a long time to converge to the global minimizer. Since we are only interested in the value of the minimum (and not of the minimizer), we may modify the stopping criterion to

$$\mu_+ - \mu_- \leq \epsilon(1 + \mu_- + \mu_-) \text{ or } \phi'(\mu_+) - \phi'(\mu_-) \leq \epsilon.$$

This second criterion retains the same complexity as the original bisection algorithm. The second condition in the criterion implies that $|\phi'((\mu_+ + \mu_-)/2)| \leq \epsilon$, which is an approximate condition for global optimality.

8.5 Maximum a posteriori models

We now consider a variation on the likelihood model, the Maximum a posteriori (MAP) model. The MAP estimation framework provides a way of incorporating prior information in the estimation process. This is particularly useful for dealing with sparse training data, for which the maximum likelihood approach may provide inaccurate estimates. The MAP estimator, denoted by p^{MAP}, maximizes the "MAP function"[33]

$$L_{\text{MAP}}(p) = L(p) + \log g_{\text{prior}}(p),$$

in which $L(p)$ is the log-likelihood function, and g_{prior} refers to the *a priori* density function of the parameter vector p.

In our case, p is a row of the transition matrix, so a prior distribution has support included in the n-dimensional simplex $\{p : p \geq 0,\ p^T\mathbf{1} = 1\}$. It is customary to choose the prior to be a Dirichlet distribution [59, 34], the density of which is of the form

$$g_{\text{prior}}(p) = K \cdot \prod_i p_i^{\alpha_i - 1},$$

in which the vector $\alpha \geq \mathbf{1}$ is given, and K is a normalizing constant. Choosing $\alpha = \mathbf{1}$ we recover the 'non-informative prior', which is the uniform distribution on the n-dimensional simplex. In that case, the MAP estimation converges to the Maximum Likelihood estimation. Hence, MAP estimation is a more general framework and Maximum Likelihood estimation is a specialization of MAP when prior information is not available.

The resulting MAP estimation problem takes the form

$$\max_p \, (f + \alpha - \mathbf{1})^T \log p \ : \ p^T \mathbf{1} = 1, p \geq 0.$$

To this problem we can associate a "MAP" region which describes the uncertainty on the estimate, via a lower bound β on the function $L_{\rm MAP}(p)$. The inner problem now takes the form

$$\phi := \max_p \, p^T v \ : \ p \geq 0, \ \ p^T \mathbf{1} = 1, \ \ \sum_j \, (f(j) + \alpha(j) - 1) \log p(j) \geq \gamma,$$

in which γ depends on the normalizing constant K appearing in the prior density function and on the chosen lower bound on the MAP function, β. We observe that this problem has exactly the same form as in the case of likelihood function, provided we replace f by $f + \alpha - \mathbf{1}$. Therefore, the same results apply to the MAP case.

8.6 Entropy Models

8.6.1 Model description

Here, we describe the uncertainty on each row of the transition matrix via an entropy constraint. Specifically we consider problem (51), with the uncertainty on the i-th row of the transition matrix P^a described via a lower bound on the entropy function relative to a given distribution q (Kullback-Leibler divergence)

$$\mathcal{U}(\beta) = \left\{ p \in \mathbf{R}^n \ : \ p^T \mathbf{1} = 1, \ p \geq 0, \ \ \sum_j p(j) \log \frac{p(j)}{q(j)} \leq \beta \right\}.$$

Here $q > 0$ is a given distribution, and $\beta > 0$ is fixed. We can chose the maximum likelihood estimate as the value of q. Together with $q > 0$, the condition $\beta > 0$ ensures that $\mathcal{U}$ has non-empty interior. (As before, we have dropped the control and row indices a and i).

We now address the inner problem (51), with $\mathcal{U} = \mathcal{U}(\beta)$ given above. We note that the set above actually equals the whole probability simplex if β is too large, specifically if $\beta \geq \max_i(-\log q_i)$, since the latter quantity is the maximum of the relative entropy function over the simplex. Thus, if $\beta \geq \max_i(-\log q_i)$, the worst-case value of $p^T v$ for $p \in \mathcal{U}(\beta)$ is equal to $v_{\max}$.

8.6.2 Dual problem

By standard duality arguments (set $\mathcal{U}$ being strictly feasible), the inner problem is equivalent to its dual:

$$\min_{\lambda > 0, \mu} \ \mu + \beta\lambda + \lambda \sum_j q(j) \exp\left(\frac{v(j) - \mu}{\lambda} - 1 \right).$$

Setting the derivative with respect to μ to zero, we obtain the optimality condition

$$\sum_j q(j) \exp\left(\frac{v(j)-\mu}{\lambda} - 1\right) = 1,$$

from which we derive

$$\mu = \lambda \log\left(\sum_j q(j) \exp\frac{v(j)}{\lambda}\right) - \lambda.$$

The optimal distribution is

$$p^* = \frac{q(j)\exp\frac{v(j)}{\lambda}}{\sum_i q(i)\exp\frac{v(i)}{\lambda}}.$$

As before, we reduce the problem to a one-dimensional problem:

$$\min_{\lambda>0} \phi(\lambda)$$

in which ϕ is the convex function:

$$\phi(\lambda) = \lambda \log\left(\sum_j q(j) \exp\frac{v(j)}{\lambda}\right) + \beta\lambda. \tag{59}$$

Perhaps not surprisingly, the above function is closely linked to the moment generating function of a random variable $\mathbf{v}$ having the discrete distribution with mass q_i at v_i.

8.6.3 A bisection algorithm

As proved in Appendix 8.16.3, the convex function ϕ in (59) has the following properties:

$$\forall\, \lambda \geq 0, \quad q^T v + \beta\lambda \leq \phi(\lambda) \leq v_{\max} + \beta\lambda, \tag{60}$$

and

$$\phi(\lambda) = v_{\max} + (\beta + \log Q(v))\lambda + o(\lambda), \tag{61}$$

in which

$$Q(v) := \sum_{j\,:\,v(j)=v_{\max}} q(j) = \textbf{Prob}\{\mathbf{v} = v_{\max}\}.$$

Hence, $\phi(0) = v_{\max}$ and $\phi'(0) = \beta + \log Q(v)$. In addition, at infinity the expansion of ϕ is

$$\phi(\lambda) = q^T v + \beta\lambda + o(1). \tag{62}$$

The bisection algorithm can be started with the lower bound $\lambda_- = 0$. An upper bound can be computed by finding a solution to the equations $\phi(0) = q^T v + \beta\lambda$, which yields $\lambda_+ = (v_{\max} - q^T v)/\beta$. By convexity, a minimizer exists in the interval $[0\ \lambda_+]$.

Note that if $\phi'(0) \geq 0$, then $\lambda = 0$ is optimal and the optimal value of ϕ is $v_{\max}$. This means that if β is too high, that is, if $\beta > -\log Q(v)$, enforcing robustness amounts to disregard any prior information on the probability distribution p. We have observed in 8.6.1 a similar phenomenon brought about by too large values of β, which resulted in a set $\mathcal{U}$ equal to the probability simplex. Here, the limiting value $-\log Q(v)$ depends not only on q but also on v, since we are dealing with the optimization problem (51) and not only with its feasible set $\mathcal{U}$.

8.7 Other Specific Models

8.8 Interval matrix model

The *interval matrix* model is

$$\mathcal{U} = \left\{ p \ : \ \underline{p} \leq p \leq \overline{p}, \ \ p^T \mathbf{1} = 1 \right\},$$

in which $p_\pm$ are given componentwise non-negative n-vectors (whose elements do not necessarily sum to one), with $p_+ \geq p_-$. This model is motivated by statistical estimates of intervals of confidence on the *components* of the transition matrix. Those intervals can be obtained by resampling methods, or by projecting an ellipsoidal uncertainty model on each component axis (see section 8.9). In what follows, we assume that $\mathcal{U}$ is not empty.

Since the inner problem

$$\phi := \max_p \ v^T p \ : \ p \geq 0, \ \ p^T \mathbf{1} = 1, \ \ \underline{p} \leq p \leq \overline{p}$$

is a linear, feasible program, it is equivalent to its Lagrange dual, which has the form

$$\phi = \min_\mu \ (\overline{p} - \underline{p})^T (\mu \mathbf{1} - v)^+ + v^T \overline{p} + \mu(1 - \overline{p}^T \mathbf{1}),$$

in which z^+ stands for the positive part of vector z. The function to be minimized is a convex piecewise linear function with break points $v_0 = 0$, $v_1, \ldots, v_n$. Since the original problem is feasible, we have $\mathbf{1}^T \underline{p} \leq 1$, which implies that the function above goes to infinity when $\mu \to \infty$. Thus, the minimum of the function is attained at one of the break points v_i ($i = 0, \ldots, n$). The complexity of this enumerative approach is $O(n^2)$, since each evaluation costs $O(n)$.

In fact one does not need to enumerate the function at all values v_i; a bisection scheme over the discrete set $\{v_0, \ldots, v_n\}$ suffices. This scheme will bring the complexity down to $O(n \log n)$.

8.9 Ellipsoidal models

Ellipsoidal models arise when second-order approximations are made to the log-likelihood function arising in the likelihood model. Specifically, we work with the following set in lieu of (52):

$$\mathcal{P}(\beta) = \left\{ P \in \mathbf{R}^{n \times n} \ : \ P \geq 0, \ \ P\mathbf{1} = \mathbf{1}, \ \ Q(P) \geq \beta \right\}, \tag{63}$$

in which $Q(P)$ is the second-order approximation to the log-likelihood function L, around the maximum-likelihood estimate F:

$$Q(P) := \beta_{\max} - \frac{1}{2} \sum_{i,j} \frac{(P(i,j) - F(i,j))^2}{F(i,j)}.$$

This set is an ellipsoid intersected by the polytope of transition matrices. Again, the projection on the space of i^{th} row variables assumes a similar shape, that of an ellipsoid intersected with the probability simplex, specifically

$$\mathcal{P}_i(\beta) = \left\{ p \ : \ p \geq 0, \ \ p^T \mathbf{1} = 1, \ \ \sum \frac{(p_i(j) - f_i(j))^2}{f_i(j)} \leq \kappa^2 \right\},$$

in which $\kappa^2 := 2(\beta_{\max} - \beta)$. We refer to this model as the *constrained ellipsoidal model*.

In the constrained likelihood case, the inner problem assumes the form

$$\max_p \ v^T p \ : \ p \geq 0, \ \ p^T \mathbf{1} = 1, \ \ \sum \frac{(p_i(j) - f_i(j))^2}{f_i(j)} \leq \kappa^2.$$

According to [66], this problem has worst-case complexity of $O(n^{3.5})$. This brings the complexity of one step of the robust Bellman recursion to $O(n^{3.5}m)$.

In statistics, it is a standard practice to further simplify the description above, by relaxing the inequality constraints $P \geq 0$ in the definition of $\mathcal{P}(\beta)$. We thus obtain the (unconstrained) *ellipsoidal* model, which leads to

$$\phi := \max_p \ v^T p \ : \ p^T \mathbf{1} = 1, \ \ \sum \frac{(p_i(j) - f_i(j))^2}{f_i(j)} \leq \kappa^2.$$

Taking the dual of the above problem, we obtain the closed-form expression

$$\phi = f_i^T v + \kappa \sqrt{\sum_j f_i(j)(v(j) - f_i^T v)^2},$$

which has $O(n)$ complexity. The robust recursion based on the unconstrained ellipsoidal model is thus $O(nm)$, the same as that of the classical Bellman recursion.

This economical computation comes at an expense, which is the possible conservatism of the worst-case value function stemming from our neglect of the non-negativity constraints on the transition matrix. Another potential problem is the fact that the ellipsoid model is symmetric around the maximum-likelihood point, which might not be realistic . In the maximum-likelihood model, the non-negativity constraints are implicit in the likelihood bound, and the model yields potentially non-symmetric (hence more realistic) estimates.

Uncertainty on the reference distribution q in entropy models. We may generalize the relative entropy models to the case when there is uncertainty on the reference distribution q.

If $\mathcal{E}$ is a set of reference distributions q, we can consider the inner problem (51), where the uncertainty set $\mathcal{U}$ replaced by one of the form

$$\mathcal{U} = \left\{ p \ : \ p \geq 0, \ \ p^T \mathbf{1} = 1, \ \ \sum_j p(j) \log \frac{p(j)}{q(j)} \leq \beta \text{ for some } q \in \mathcal{E} \right\}.$$

Using the same steps as before, the inner problem reduces to

$$\max_{q\in\mathcal{E}} \min_{\lambda>0} \lambda \log \left(\sum_j q(j) \exp \frac{v(j)}{\lambda} \right) + \beta\lambda.$$

This problem is very easy if $\mathcal{E}$ is a box (hyperrectangle) or an ellipsoid parallel to the coordinate axes. For example, assume that $\mathcal{E}$ assumes the form we encountered in the case of ellipsoidal models, specifically $\mathcal{E} = \mathcal{P}$, where $\mathcal{P}$ is given by (63). Then we obtain

$$\min_{\lambda>0} \lambda \log \left(\sum_j f(j) \exp \frac{v(j)}{\lambda} + \kappa \sqrt{\sum_j f(j)(\exp \frac{v(j)}{\lambda} - f^T \exp \frac{v}{\lambda})^2} \right) + \beta\lambda.$$

A bisection algorithm similar to the ones described earlier can be applied to this modified problem.

8.10 Example: Robust Aircraft Routing

We consider the problem of routing an aircraft whose path is obstructed by stochastic obstacles, representing storms. In practice, the stochastic model must be estimated from past weather data. This makes this particular application a good illustration of our method.

8.11 The nominal problem

In [48], we introduced an MDP representation of the problem, in which the evolution of the storms is modelled as a *perfectly* known stationary Markov chain. The term nominal here refers to the fact that the transition matrix of the weather Markov chain is not subject to uncertainty. The goal is to minimize the expected delay (flight time). The weather process is a fully observable Markov chain: at each decision stage (every 15 minutes in our example), we learn the actual state of the weather.

The airspace is represented as a rectangular grid. The state vector comprises the current position of the aircraft on the grid, as well as the current states of each storm. The action in the MDP corresponds to the choice of nodes to fly towards, from any given node. There are k obstacles, represented by a Markov chain with a $2^k \times 2^k$ transition matrix. The transition matrix for the routing problem is thus of order $N2^k$, where N is the number of nodes in the grid.

We solved the MDP via the Bellman recursion [48]. Our framework avoids the potential "curse of dimensionality" inherent in generic Bellman recursions, by considerable pruning of the state-space and action sets. This makes the method effective for up to a few storms, which corresponds to realistic situations. For more details on the nominal problem and its implementation, we refer the reader to [48].

In the example below, the problem is two-dimensional in the sense that the aircraft evolves at a fixed altitude. In a coordinate system where each unit is equal to 1 Nautical Mile, the aircraft is initially positioned at $(0, 0)$ and the destination point is at $(360, 0)$. The velocity of the aircraft is fixed at 480 n.mi/hour. The airspace is described by a

rectangular grid with $N = 210$ nodes, with edge length of 24 n.mi. There is a possibility that a storm might obstruct the flight path. The storm zone is a rectangular space with the corner points at $(160, 192), (160, -192), (168, 192)$ and $(168, -192)$ (figure 23).

Since there is only one potential storm in the area, storm dynamics is described by a 2×2 transition matrix P_{weather}. Together with $N = 210$ nodes, this results in a state-space of total dimension 420. By limiting the angular changes in the heading of the aircraft, we can prune out the action space and reduce its cardinality at each step to $m = 4$. This implies that the transition matrices are very sparse; in fact, they are sparse, affine functions of the transition matrix P_{weather}. Sparsity implies that the nominal Bellman recursion only involves 8 states at each step.

8.12 The robust version

In practice, the transition matrix P_{weather} is estimated from past weather data, and thus it is subject to estimation errors.

We assume a likelihood model of uncertainty on this transition matrix. This results in a likelihood model of uncertainty on the state transition matrix, which is as sparse as the nominal transition matrix. Thus, the effective state pruning that takes place in the nominal model can also take place in the robust counterpart. In our example, we chose the numerical value

$$P_{\text{weather}} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

for the maximum-likelihood estimate of P_{weather}.

The likelihood model involves a lower bound β on the likelihood function, which is a measure of the uncertainty level. Its maximum value $\beta_{\max}$ corresponds to the case with no uncertainty, and decreasing values of β correspond to higher uncertainty level. To β, we may associate a measure of uncertainty that is perhaps more readable: the *uncertainty level*, denoted U_L, is defined as a percentage and its complement $1 - U_L$ can be interpreted as a probabilistic confidence level in the context of large samples. The one-to-one correspondence of U_L and β is precisely described in Appendix 8.16.4.

In figure 24, we plot U_L against decreasing values of the lower bound on the log-likelihood function (β). We see that $U_L = 0$, which refers to a complete certainty of the data, is attained at $\beta = \beta_{max}$, the maximum value of the likelihood function. The value of U_L decreases with β and reaches the maximum value, which is 100%, at $\beta = -\infty$ (not drawn in this plot). Point to be noted: the rate of increase of U_L is maximum at $\beta = \beta_{max}$ and increases with β.

8.13 Comparing robust and nominal strategies

In figure 25, we compare various strategies: we plot the relative delay , which is the relative increase (in percentage) in flight time with respect to the flight time corresponding to the most direct route (straight line), against the negative of the lower bound on the likelihood function β.

We compare three strategies. The *conservative* strategy is to avoid the storm zone altogether. If we take $\beta = \beta_{max}$, the uncertainty set becomes a singleton ($U_L = 0$) and hence we obtain the solution computed via the classical

Bellman recursion; this is referred to as the *nominal* strategy. The *robust* strategy corresponds to solving our robust MDP with the corresponding value of β.

The plot in figure 25 shows how the various strategies fare, as we decrease the bound on the likelihood function β. For the nominal and the robust strategies, and a given bound β, we can compute the worst-case delay using the recursion (50), which provides the worst-case value function.

The conservative strategy incurs a 51.5% delay with respect to the flight time corresponding to the most direct route. This strategy is independent of the transition matrix, so it appears as a straight line in the plot. If we know the value of the transition matrix exactly, then the nominal strategy is extremely efficient and results in a delay of 8.02% only. As β deviates from β_{max}, the uncertainty set gets bigger. In the nominal strategy, the optimal value is very sensitive in the range of values of β close to β_{max}: the delay jumps from 8% to 25% when β changes by 7.71% with respect to $\beta_{\max}$ (the uncertainty level U_L changes from 0% to 5%). In comparison, the relative delay jumps by only 6% with the robust strategy. In both strategies, the slope of the optimal value with respect to the uncertainty is almost infinite at $\beta = \beta_{max}$, which shows the high sensitivity of the value function with respect to the uncertainty.

We observe that the robust solution performs better than the nominal solution as the estimation error increases. The plot shows an average of 19% decrease in delay with respect to the nominal strategy when uncertainty is present. Further, the nominal strategy very quickly reaches delay values comparable to those obtained with the conservative strategy, as the uncertainty level increases. In fact, the conservative strategy even outperforms the nominal strategy at $\beta = -1.84$, which corresponds to $U_L = 69.59\%$. In this sense, even for moderate uncertainty levels, the nominal strategy defeats its purpose. In contrast, the robust strategy outperforms the conservative strategy by 15% even if the data are very uncertain ($U_L = 85\%$).

In summary, when there is no error in the estimation, both nominal and robust algorithms provide a strategy that produces 43.3% less delay than the conservative strategy,. However, with the presence of even a moderate estimation error, the robust strategy performs much better than the conservative strategy, whereas the nominal MDP strategy cannot produce a much better result.

Nominal and robust strategies have similar computational requirements. In our example, with a simple Matlab implementation on a standard PC, the running time for the nominal algorithm was about 4 seconds, and the robust version took on average 4 more seconds to solve.

8.14 Inaccuracy of uncertainty level

The previous comparison assumes that, in the robust case, we are able to estimate exactly the precise value of the uncertainty level U_L (or the bound on the likelihood function β). In practice, this parameter also has to be estimated. Hence the question: how sensitive is the robust approach with respect to inaccuracies in the uncertainty level U_L?

To answer this question in our particular example, we assume that a guess U_L^0 on the uncertainty level is available, and examined how the corresponding robust solution would behave if it was subject to uncertainty with level above or below the guess.

In figure 26, we compare various strategies. In each strategy, we guess a desired level of accuracy (U_L^0) on the data and calculate a corresponding likelihood bound β^0. We choose the optimal action using our robust MDP algorithm

applied with this bound. Keeping the resulting policy fixed, we then compute the relative delay with the various values of β. In figure 26, we plot the relative delays against $-\beta$ for the strategies where the uncertainty levels were guessed as 15% and 55%.

Not surprisingly, the relative delay of a strategy attains its minimum value when β (U_L) is accurately predicted. For values of β above or below its guessed value, the delay increases. We note that it is only for very small uncertainty levels (within $.995\%$ of $\beta_{\max}$) that the nominal strategy performs better than the robust strategy with imperfect prediction of β (U_L).

We define R_{U_L} as the range of the actual U_L in percentage terms where the robust strategy (with imperfect prediction of U_L) performs worse than nominal strategy. In figure 27, we show R_{U_L} against the guessed value, U_L^0. The plot clearly shows that R_{U_L} remains less than 1% with varying predicted U_L^0.

Our example shows that if we predict the uncertainty level inaccurately in order to obtain a robust strategy, the nominal strategy will outperform the robust strategy only if the actual uncertainty level U_L is less than 1%. For any higher value of the uncertainty level, the robust strategies outperform the nominal strategy, by an average of 13%. Thus, even if the uncertainty level is not accurately predicted, the robust solution outperforms the nominal solution significantly.

8.15 Concluding remarks

We have considered uncertainty models on the transition matrix that are statistically accurate and give rise to a very moderate increase in computational cost. All the models, (except the interval matrix model), considered here give rise to inner problems with worst-case complexity less than $O(n)$. With these models, the total cost of one step of the robust Bellman recursion is thus $O(mn)$ (m is the number of actions). This has the same same complexity as the classical recursion, which has complexity of $O(mn)$. In the interval matrix model, the the worst-case complexity is $O(mn \log n)$.

From the point of view of statistical accuracy, the likelihood or entropy models are certainly preferable to the ellipsoid or interval models: these models take into account sign constraints, possibly asymmetric uncertainty around the maximum-likelihood or minimum relative entropy point, in contrast to the ellipsoidal and box uncertainty models that are possibly crude approximations to the above models.

We have shown in a practical path planning example the benefits of using a robust strategy instead of the classical optimal strategy; even if the uncertainty level is only crudely guessed, the robust strategy yields a much better expected flight delay.

Acknowledgments

The authors would like to thank Antar Bandyopadhyay, Giuseppe Calafiore, Ashwin Ganesan, Jianghai Hu, Mikael Johansson, Rupak Majumdar, Andrew Ng, Stuart Russell, Shankar Sastry, and Pravin Varaiya for interesting discussions and comments. The authors are specially grateful to Dimitri Bertsimas for pointing out an important mistake in the earlier version of this work.

8.16 Appendix

8.16.1 Proof of the robust Bellman recursion

In this section, we prove that the stochastic game (48) can be solved using the robust Bellman recursion (49). Our proof is based on transforming the original problem into a term-based zero-sum game, and applying a result by Nowak [52] that applies to such games.

We begin by augmenting the state space $\mathcal{X}$ with states of the form (i, a), where $i \in \mathcal{X}$ and $a \in \mathcal{A}$. The augmented state-space is thus $\mathcal{X}^{\text{aug}} := \mathcal{X} \bigcup (\mathcal{X} \times \mathcal{A})$. We now define a new two-player game on this augmented state-space, where decisions are taken not only at time t, $t \in T = \{0, 1, \ldots, N\}$, but also at intermediate times $t + 1/2$, $t \in T$.

In the first step, from t to $t + 1/2$, if the controller action is a_t, states of the form i make a transition to states of the form (i, a_t) with probability one, and all other states stay the same with probability one. Here, the opponent is idle. The cost incurred by this step is the cost of the original problem, $c_t(i, a_t)$, if we start from state i, and zero otherwise.

In the second step, from $t + 1/2$ to $t + 1$, the controller stands idle while the opponent acts as follows. The states of the form (i, a) make a transition to states of the form j with probabilities given by the vector p_i^a, in which p_i^a is chosen freely by the opponent in the set $\mathcal{P}_i^a$; all the other states stay the same with probability one. There is no cost incurred at this stage.

Clearly, starting at time t in state i, and with a controller action a_t, we end up in the state j at time $(t + 1)$ with probability $P^{a_t}(i, j)$. Since incurred costs are the same, our new game is equivalent to the original game. In addition, the new game is a term-based zero-sum game, since the controller and the opponent act alternatively, in independent fashion at each time step.

Nowak's result provides a Bellman-type recursion to solve the problem of minimizing the worst-case (maximum) expected cost of a term-based zero-sum game, when both players follow randomized policies that are restricted to given state-dependent convex subsets of the probability simplex. In our new game, the opponent's choice of a vector p_i^a within $\mathcal{P}_i^a$ at the second step, can be interpreted as a choice of a randomized policy over the convex, state-dependent set $\mathcal{B}((i, a)) := \mathcal{P}_i^a$. (Here, the deterministic actions of the opponent correspond to the vertices of the probability simplex of $\mathbf{R}^n$.) Hence, the results due to Nowak [52] apply.

In the case when one player (say, the first) acts deterministically, with state-independent, finite action set $\mathcal{A}$, the recursion for the optimal value function V in state s can be written as

$$V_t(s) \quad = \quad \min_{a \in \mathcal{A}} \max_{\mathbf{b} \in \mathcal{B}(s)} \mathrm{E}_{\mathbf{b}} \left(c_t(s, a, b) + \sum_{s'} P^{ab}(s, s') V_{t+1}(s') \right), \tag{64}$$

in which the notation a, b refers to actions of the minimizing and maximizing player respectively, P^{ab} is the corresponding transition matrix, c_t is the cost function, $\mathbf{b}$ refers to a particular randomized action that is freely chosen by the opponent within the state-dependent convex compact set $\mathcal{B}(s)$, and $\mathrm{E}_{\mathbf{b}}$ is the corresponding expectation operator.

Let us detail how applying the above recursion to our game yields our result.

Denote by $V_t(s)$ the value function of the game at time t in state $s \in \mathcal{X}^{\text{aug}}$. We first update this value function from time $t+1$ to $t+1/2$. The controller is idle, but the opponent is allowed to chose a randomized policy from a state-dependent convex-compact set. If the state is (i,a), this set is $\mathcal{B}((i,a)) = \mathcal{P}_i^a$, and the value function is updated as

$$V_{t+\frac{1}{2}}((i,a)) \quad = \quad \max_{p \in \mathcal{P}_i^a} \left(\sum_{j=1}^{n} p(j) V_{t+1}(j) \right), \tag{65}$$

in which we make use of the fact that incurred costs are zero in this step. To update the value function from $t+1/2$ to t, we use the fact that the opponent is idle. For $i = 1, \ldots, n$, the value function is updated as

$$V_t(i) \quad = \quad \min_{a \in \mathcal{A}} \left(c_t(i,a) + V_{t+\frac{1}{2}}((i,a)) \right). \tag{66}$$

Combining (66) and (65) ends our proof.

8.16.2 Properties of function ϕ of section 8.4.3

Here, we prove two properties of the function ϕ involved in the bisection algorithm of section 8.4.3. For simplicity of notation, we assume that there is an unique index i^* achieving the maximum in $v_{\max}$, that is, $v(i^*) = v_{\max}$.

We first show that $\phi(\mu) \to v_{\max}$ as $\mu \to v_{\max}$. We have

$$\lambda(\mu) = \frac{\mu - v(i^*)}{f(i^*)} + o(\mu - v(i^*)).$$

We then express $\phi(\mu)$ as

$$\begin{aligned} \phi(\mu) = \mu - \lambda(\mu) \left(1 + \beta - \beta_{\max} + \log \lambda(\mu) - \sum_{j \neq i^*} f_j \log(\mu - v_j) \right) \\ - \lambda(\mu) f(i^*) \log(\mu - v(i^*)). \end{aligned}$$

The second term (first line) vanishes as $\mu \to v_{\max}$, since $\lambda(\mu) \to 0$ then. In view of the expression of $\lambda(\mu)$ above, the last term (second line) behaves as $(\mu - v(i^*)) \log(\mu - v(i^*))$, which also vanishes.

Next we prove that $\phi'(\mu) \to -\infty$ as $\mu \to v_{\max}$. We obtain easily

$$\frac{d\lambda(\mu)}{d\mu} = \frac{\sum_j \dfrac{f(j)}{(\mu - v(j))^2}}{\left(\sum_j \dfrac{f(j)}{\mu - v(j)} \right)^2} \to \frac{1}{f(i^*)} \text{ when } \mu \to v(i^*).$$

We then have

$$\begin{aligned}\frac{\partial h}{\partial \lambda}(\lambda(\mu),\mu) &= \sum_j \log \frac{\lambda(\mu) f(j)}{\mu - v(j)} - \beta \\ &= \log \frac{\lambda(\mu) f(i^*)}{\mu - v(i^*)} + \sum_{j \neq i^*} \log \frac{\lambda(\mu) f(j)}{\mu - v(j)} - \beta \\ &= \log(1 + o(1)) + (n-1)\log \lambda(\mu) + \sum_{j \neq i^*} \log \frac{f(j)}{\mu - v(j)} - \beta \\ &\to -\infty \text{ as } \mu \to v(i^*).\end{aligned}$$

Also, by definition of $\lambda(\mu)$, we have $\partial h/\partial \mu(\lambda(\mu),\mu) = 0$. The proof is achieved with

$$\phi'(\mu) = \frac{\partial h}{\partial \mu}(\lambda(\mu),\mu) + \frac{\partial h}{\partial \lambda}(\lambda(\mu),\mu)\frac{d\lambda(\mu)}{d\mu}.$$

8.16.3 Properties of function ϕ of section 8.6.3

In this section, we prove that the function ϕ defined in (59) obeys properties (60), (61) and (62).

First, we prove (61). If $v(j) = v_{\max}$ for every j, the result holds, with $Q(v) = Q(v_{\max}\mathbf{1}) = 1$. Assume now that there exist j such that $v(j) < v_{\max}$. We have

$$\begin{aligned}\phi(\lambda) &= \lambda \log \left(e^{v_{\max}/\lambda} \textstyle\sum_j q(j) \exp(\frac{v(j) - v_{\max}}{\lambda}) \right) + \beta\lambda \\ &= v_{\max} + \beta\lambda + \lambda \log \left(\sum_{j:v(j)=v_{\max}} q(j) + \sum_{j:v(j)<v_{\max}} q(j) \exp(\frac{v(j) - v_{\max}}{\lambda}) \right) \\ &= v_{\max} + \beta\lambda + \lambda \log \left(Q + O(e^{-t/\lambda}) \right) \\ &= v_{\max} + (\beta + \log Q)\lambda + O(\lambda e^{-t/\lambda}),\end{aligned}$$

in which $t = v_{\max} - v_s > 0$, in which v_s is the largest $v(j) < v_{\max}$. This proves (61).

From the expression of ϕ given in the second line above, we immediately obtain the upper bound in (60).

The expansion of ϕ at infinity provides

$$\begin{aligned}\phi(\lambda) &= \beta\lambda + \lambda \log \left(\textstyle\sum_j q(j)(1 + \frac{v(j)}{\lambda} + o(\lambda)) \right) \\ &= q^T v + \beta\lambda + o(1),\end{aligned}$$

which proves (62). The lower bound in (60) is a direct consequence of the concavity of the $\log$ function.

8.16.4 Calculation of β for a Desired Confidence Level

In this section, we describe the one-to-one correspondence between a lower bound on the likelihood function, as used in section 8.4, with a desired level of confidence $(1 - U_L)$ on the transition matrix estimates. This correspondence is valid for asymptotically large samples only but can serve as a guideline to choose β.

First, we define a vector $q_i = [P(i,1), \ldots, P(i,n-1)]^T, \forall i = 1, \ldots, n$ and $\theta = [q_1, \ldots, q_n]^T \in \mathbf{R}^{n(n-1)}$, in which P is the transition matrix that we want to estimate. Hence, $P(i,j) = \theta_{ij} = \theta((n-1)^2 i + j)\ \forall\ 1 \leq i \leq n, 1 \leq j \leq (n-1)$. Provided some regularity conditions hold [36], it is possible to make Laplace approximation of the Likelihood function and we can make the following asymptotic statement about the distribution of θ: precisely, that θ is normally distributed with the mean given by $\hat{\theta}_{ij} := F(i,j),\ 1 \leq i \leq n, 1 \leq j \leq (n-1)$ and covariance matrix $I(\theta)$ (Fisher Information matrix) given by

$$I(\theta)_{pq} = E_\theta \left(-\frac{\partial^2}{\partial\theta_p \partial\theta_q} l(\theta) \right)\ \forall\ p, q = 1, \ldots, n(n-1), \tag{67}$$

in which $l(.) = \log(L(.))$ is the log-likelihood function.

We can approximate $I(\theta)$ with the observed information matrix, which is meaningful in the neighborhood of $\hat{\theta}$. The equation of the observed information matrix is given by

$$I_o(\theta)_{pq} = -\frac{\partial^2}{\partial\theta_p \partial\theta_q} l(\theta)\ \forall\ p, q = 1, \ldots, n(n-1), \tag{68}$$

in which $\frac{\partial^2}{\partial\theta_p \partial\theta_q} l(\theta)$ can be shown to be

$$\frac{\partial^2}{\partial\theta_p \partial\theta_q} l(\theta) = \begin{cases} -\frac{F(p,q)+F(q,n)}{F(p,n)F(p,q)}, & \text{if } p, q \text{ correspond to the elements in a same row in } P \text{ and } p = q, \\ -\frac{1}{F(p,n)}, & \text{if } p, q \text{ correspond to the elements in a same row in } P \text{ and } p \neq q, \\ 0, & \text{if } p \text{ and } q \text{ correspond to the elements in different rows in } P. \end{cases} \tag{69}$$

This is true for large number of sample [35]. We further define, $H := I_o(\theta)$. Then the parameter β is chosen to be the smallest such that, under the probability distribution $N(\hat{\theta}, (H)^{-1})$, the set,

$$\xi_\beta = \{\theta : \tilde{l}(\theta) \geq \beta\}, \tag{70}$$

in which $\tilde{l}(\theta)$ is the quadratic approximation to $l(\theta)$ around $\theta = \hat{\theta}$, that is,

$$\tilde{l}(\theta) = \beta_{max} - \frac{1}{2}(\theta - \hat{\theta})^T H (\theta - \hat{\theta}), \tag{71}$$

has the probability larger than a threshold $(1 - U_L)$, where (say) $U_L = 15\%$ in order to obtain the 85% confidence level.

It turns out that, we can solve for such a β explicitly,

$$(1 - U_L) = F_{\chi^2_{n(n-1)}}(2(\beta_{max} - \beta)), \tag{72}$$

in which $F_{\chi^2_{n(n-1)}}(.)$ is the cumulative χ^2 distribution with the degrees of freedom $n(n-1)$, which can be approximated by the following equation [37]

$$F_{\chi^2_{n(n-1)}}(2(\beta_{max}-\beta)) \approx \Phi(z) - \frac{\sqrt{2}}{3\sqrt{n(n-1)}}(z^2-1)\phi(z) \approx U_L, \tag{73}$$

in which, $z = \frac{2(\beta_{max}-\beta)-n(n-1)}{\sqrt{2n(n-1)}}$, $\phi(z) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2}$ and $\Phi(z) = \int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}p^2}dp$ is the standard normal cumulative density function.

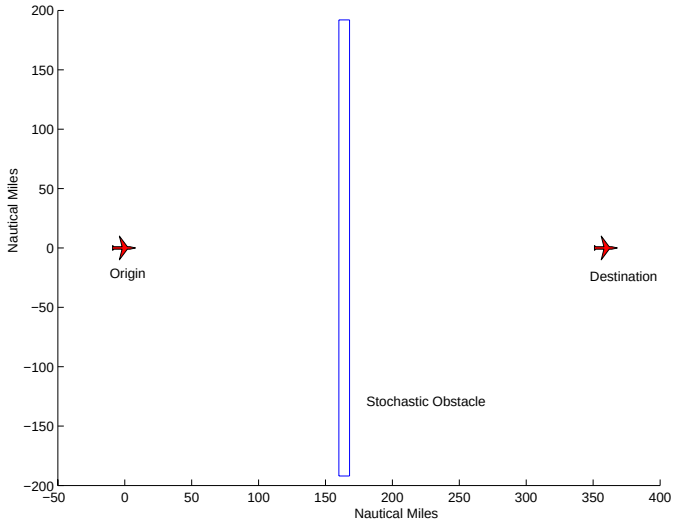


Figure 23: Aircraft Path Planning Scenario.

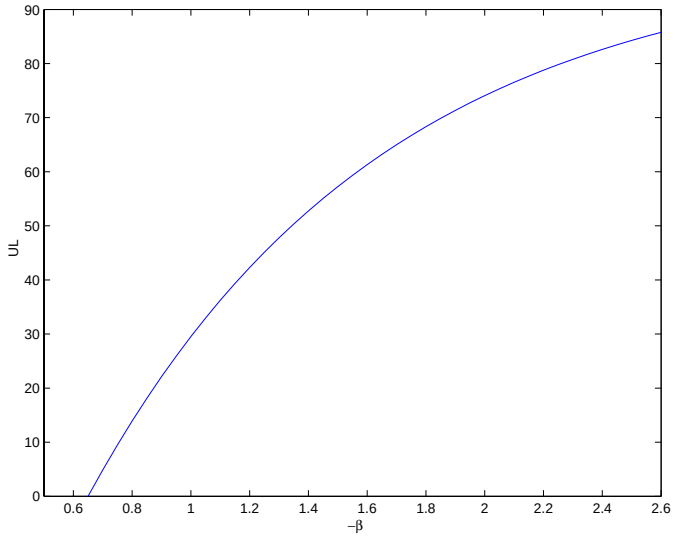


Figure 24: $-\beta$ (negative lower bound on the log-likelihood function) vs U_L (Uncertainty Level (in $\%$) of the Transition Matrices).

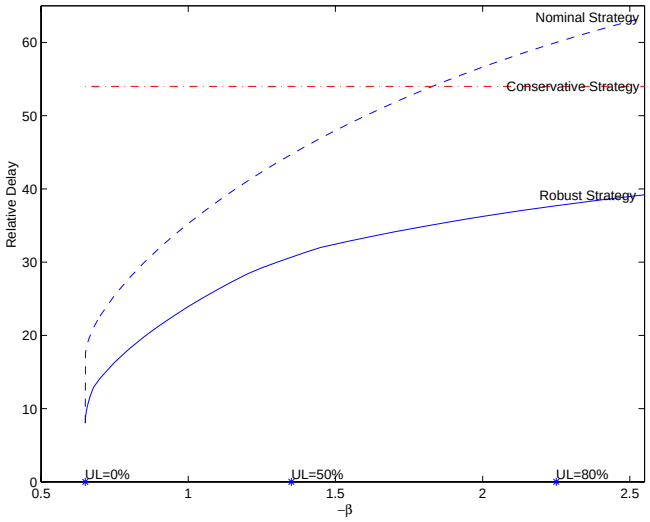


Figure 25: Optimal value vs. uncertainty level (negative lower bound on the log-likelihood function), for both the classical Bellman recursion and its robust counterpart.

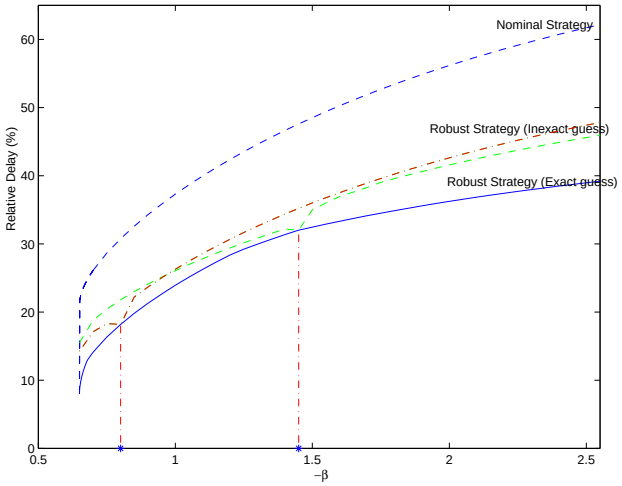


Figure 26: Optimal value vs. uncertainty level (negative lower bound on the log-likelihood function), for the classical Bellman recursion and its robust counterpart (with exact and inexact predictions of the uncertainty level $U_L = 15\%, 55\%$).

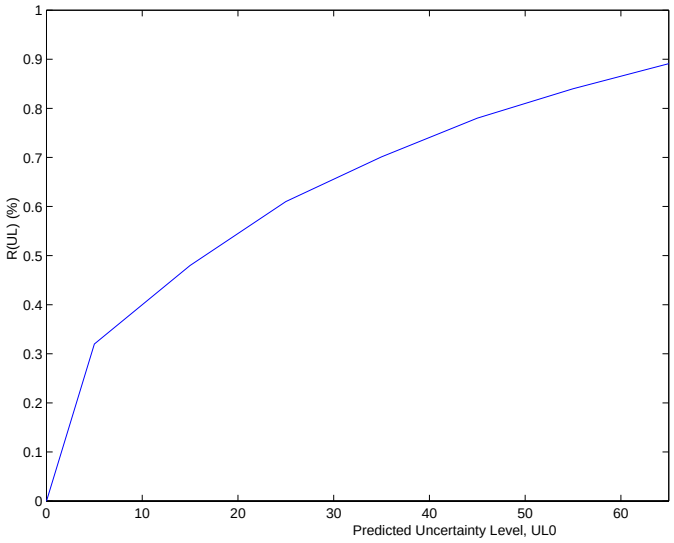


Figure 27: Predicted uncertainty level U_L^0 vs. R_{U_L}, which is the range of the actual uncertainty level U_L over which the nominal strategy performs better than a robust strategy computed with the imperfect prediction U_L^0.

9 Module 7: Flexible team formation

We discuss a strategic planning problem of allocating resources to groups of tasks organized in successive stages. Each stage is characterized by a set of survival rates whose value is imprecisely known. The goal is to allocate the resources to the tasks (i.e. to form 'teams') by dynamically re-organizing the teams at each stage, while minimizing a cost objective over the whole stage horizon. A modelling framework is proposed, based on linear programming with adjustable variables. The resulting 'uncertain linear program' is subsequently solved using the sampled scenarios randomized technique. [17]

9.1 Problem Statement

We start by describing the basic model. Consider Figure 28, and suppose that a total amount C of a single type of resource is available at an initial stage. These resources should be committed to a series of tasks, which are organized in successive stages, $s = 1, \dots, N$. For instance, at the initial stage, a team $x(1) = [x_1(1) \cdots x_m(1)]^T$ is formed, where $x_i(1)$ denotes the amount of resource allocated for the i-th task in the first stage.[18] In general, we denote by $x(s) = [x_1(s) \cdots x_m(s)]^T$ the composition of the team that is allocated for the s-th stage, and we assume that each stage is composed of a fixed number m of tasks.

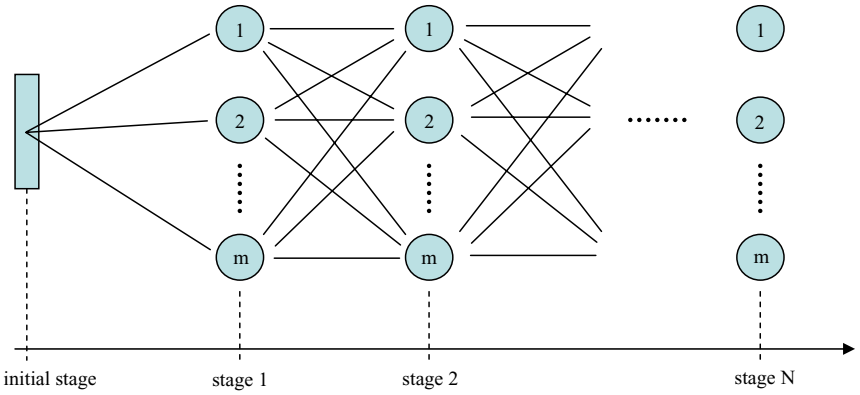


Figure 28: Multi-stage resource allocation model.

The composition of the team $x(s+1)$ (i.e. the team that should go to stage $s+1$) is decided just after the team $x(s)$ has engaged and completed the s-th stage. In our basic model, when a team engages a stage, it incurs some losses, which are described by a matrix $R(s) = \mathbf{diag}\,(r(s))$ of *survival rates* $r(s) = [r_1(s) \cdots r_m(s)]^T$. If we denote by $x(s_+)$ the composition of the team $x(s)$ just after it completed the engagement with stage s, then we have

$$x(s_+) = R(s)x(s).$$

Based on the outcome of stage s, at 'time' s_+ we have the opportunity of re-adjusting the composition of team, i.e. we can decide to re-allocate resources from one task to another, before attacking stage $s+1$. This means that the

[17] The research reported here was conducted by G. Calafiore, L. El Ghaoui and A. Nilim.

[18] The "amount" of resource may be constrained to be an integer, as discussed later.

composition of the team attacking stage $s+1$ is given by

$$x(s+1) = R(s)x(s) + u(s), \tag{74}$$

in which $u(s) = [u_1(s) \cdots u_m(s)]^T$ is the decision vector of resource re-allocation at stages $s = 0, \ldots, N-1$, and we set $x(0) \doteq 0$. Notice that $u_i(s) > 0$ means that more resources are committed for the i-th task, while $u_i(s) < 0$ means that the resources are withdrawn from this task.

Our goal is to determine the allocations $u(s)$, $s = 0, \ldots, N-1$, such that a certain cost objective is minimized over the entire stages horizon, and suitable constraints are satisfied. The problem constraints and objective are specified in the next section.

9.2 Constraints and optimization objective

Assume first that the stage survival rates $r(s)$, $s = 1, \ldots, N$ are exactly known in advance, and consider the dynamics of the team composition (74),

$$x(s+1) = R(s)x(s) + u(s), \quad x(0) = 0 \tag{75}$$

where $u(s) \in \mathbb{R}^m$, $s = 0, \ldots, N-1$ are the decision variables. We must impose the following physical constraints on the problem.

1. Total resources constraint.
$$\mathbf{1}^T u(0) \leq C, \tag{76}$$
in which $\mathbf{1}$ denotes a vector of ones. The initial assignment should not exceed the total availability of resources, C.

2. Conservation constraints.
$$\mathbf{1}^T u(s) = 0, \quad s = 1, \ldots, N-1. \tag{77}$$
At each stage (except for the initial one $s = 0$) the net sum of the exchanged resources must be zero.

3. Team composition constraints.
$$x_L(s) \leq x(s) \leq x_U(s), \quad s = 1, \ldots, N. \tag{78}$$
At each stage, the resources assigned to each task should remain between a-priori fixed lower limit $x_L(s)$ (for instance $x_L(s) = 0$), and upper limit $x_U(s)$. Notice that (78) are linear constraints on the decision variable $U = [u^T(0), \ldots, u^T(N-1)]^T$, that can be explicitly expressed in the form
$$x_L(s) \leq \Phi(s)U \leq x_U(s), \quad s = 1, \ldots, N, \tag{79}$$
in which we define, for $s = 1, \ldots, N$,
$$\Phi(s) \doteq \left[R^{(s-1,1)} \; R^{(s-1,2)} \; \cdots \; R^{(s-1,s-1)} \; I_m \mid 0_{m,m} \; \cdots \; 0_{m,m} \right], \tag{80}$$
$$R^{(s-1,i)} \doteq R(s-1)R(s-2)\cdots R(i), \quad \text{for } i = 1, \ldots, s-1, \tag{81}$$
(notice that, when forming $\Phi(s)U$, the left part of $\Phi(s)$ multiplies the decision variables $u(0), \ldots, u(s-1)$, while the zero part of $\Phi(s)$ multiplies $u(s), \ldots, u(N-1)$).

Optimization objective. When transferring resources from one task to another we incur a 'transition cost,' that we assume to be proportional to the amount of the transferred resources, regardless of the sign. The optimization objective is to minimize the total transition cost accumulated over the stages horizon.

We assume that $W(s) \in \mathbb{R}^{m,m} \geq 0$, $s = 0, \ldots, N-1$, are given diagonal matrices that weight the transition costs for the different tasks at the different stages, and therefore the total cost is expressed as

$$J = \sum_{s=0}^{N-1} \|W(s)u(s)\|_1 = \|\tilde{W}U\|_1, \tag{82}$$

in which $\tilde{W} \doteq \textbf{diag}\,(W(0), \ldots, W(N-1))$, and the above norm is the usual ℓ_1 vector norm, $\|x\|_1 = \sum |x_1|$.

Notice that minimizing J subject to certain constraints is equivalent to minimizing a slack variable γ subject to the original constraints, plus the constraint $J \leq \gamma$ (epigraphic form). In turn, this latter constraint can be expressed as a set of linear inequalities in the decision variable U, introducing a vector $z \in \mathbb{R}^{Nm}$ of additional slack variables:

$$\|\tilde{W}U\|_1 \leq \gamma \quad \Leftrightarrow \quad \begin{cases} -z \leq \tilde{W}U \leq z \\ \sum_{i=1}^{Nm} z_i \leq \gamma. \end{cases} \tag{83}$$

Remark 1. In the approach above, transitions are penalized irrespective of the source-destination task pair, meaning that the cost is sensitive to the net resource reallocation to task i, $u_i(s)$, but does not take into account from which of the other tasks these resources are drawn. It could instead be of interest to attribute different transition costs to different source-destination pairs. This could be taken into account as follows. Denote by $d_{ij}(s)$ the resource amount (positive or negative) to be transferred from task j to task i, $i \neq j = 1, \ldots, m$, before engaging stage s. Then, we write $u_i(s) = \sum_{j \neq i} d_{ij}(s)$, or in vector notation

$$u(s) = D(s)\mathbf{1}, \tag{84}$$

in which $[D(s)]_{ij} = d_{ij}(s)$ is a skew-symmetric matrix. Equation (84) represents a breakdown of the total transferred amounts $u(s)$ into the individual components $d_{ij}(s)$. We can therefore add the variables $d_{ij}(s)$ to the problem, and enforce the equality constraints (84), for $s = 1, \ldots, N-1$. Subsequently, the $d_{ij}(s)$ variables are inserted in the cost, by substituting to each term proportional to $|u_i(s)|$ in (82), a term proportional to a positive linear combination of $|d_{i1}(s)|, \ldots, |d_{im}(s)|$.

From the discussion in this section, we conclude that the basic resource allocation problem is expressed as a standard linear program in the variables U, ξ, γ which can be solved with great efficiency:

$$\min_{U,z,\gamma} \gamma \text{ subject to: } (76), (77), (79), (83). \tag{85}$$

Remark 2 (Integer solutions). Although in some applications it can be reasonable to allocate fractional resources to tasks (consider for instance money as a resource, and different assets as the tasks), in some other applications the resources to be allocated must be integer multiples of a some type of unit. This is for instance the case when the resources are mobile agents such as robots, UAVs, etc. In this situation, the correct problem formulation would be in the form of an integer linear program. However, due to computational difficulties in dealing with integer programs, in this report we do not use this formulation. Instead, when we know in advance that the resulting optimal solution will need to be approximated by an integer one, we introduce an 'immunization' technique that guarantees the satisfaction of constraints against all possible approximation errors. This technique is discussed in Section 9.4.

9.3 Multiple resources allocation

We next briefly describe how the basic allocation model previously discussed can be extended to deal with multiple types of resources. We hence assume hereafter that at stage $s = 0$ we have n different types of resources that should be allocated to the m tasks at stage $s = 1$, and subsequently re-organized dynamically. We denote by C_k, $k = 1, \dots, n$ the total availability of the k-th resource at the initial stage, and we let $x(s) \in \mathbb{R}^{mn}$ be the vector describing the composition of the team that is sent to stage s, $s = 1, \dots, N$. In particular, $x(s)$ is now divided into m blocks

$$x(s) \doteq \begin{bmatrix} x_1(s) \\ \vdots \\ x_m(s) \end{bmatrix},$$

in which each block $x_i(s) \in \mathbb{R}^n$, $i = 1, \dots, m$ is of the form

$$x_i(s) \doteq \begin{bmatrix} x_i^{(1)}(s) \\ \vdots \\ x_i^{(n)}(s) \end{bmatrix},$$

in which $x_i^{(k)}(s)$ denotes the amount of resource of type k that is allocated to the i-th task of stage s. The decision vector of resource re-allocations is partitioned similarly as

$$u(s) \doteq \begin{bmatrix} u_1(s) \\ \vdots \\ u_m(s) \end{bmatrix},$$

where

$$u_i(s) \doteq \begin{bmatrix} u_i^{(1)}(s) \\ \vdots \\ u_i^{(n)}(s) \end{bmatrix},$$

and $u_i^{(k)}(s)$ denotes the amount of resource of type k that we decide (upon completion of stage s) to add or subtract to the i-th task. With this notation, the team dynamics retain the structure (74)

$$x(s+1) = R(s)x(s) + u(s)$$

where the survival rate matrix $R(s) \in \mathbb{R}^{mn,mn}$ is block-partitioned conformably to $x(s), u(s)$. The total resources constraint (76) now reads

$$(\mathbf{1}_m^T \otimes e^T(n,k))u(0) \leq C_k, \quad k = 1, \dots, n,$$

in which $e^T(n,k)$ denotes a vector in $\mathbb{R}^n$ with all zeros, except for the k-th component, which is set to one. Similarly, the conservation constraints (77) are now expressed as

$$(\mathbf{1}_m^T \otimes e^T(n,k))u(s) = 0, \quad k = 1, \dots, n; \quad s = 1, \dots, N-1.$$

Basically, all the rest of the problem model and solution goes through in the same way as described for the basic problem with a single resource.

9.4 Dealing with integer approximations

As discussed in Remark 2, in some applications we need to deal with integer quantities in problem (85). In this situation, both the team composition $x(s)$ and the re-allocations $u(s)$ must be integers. This, however, is in contrast with the dynamic model (75), since $R(s)$ is real (its elements are in fact probabilities of survival), and therefore $x(s+1)$ will result to be real, even if $x(s), u(s)$ are integer vectors. One idea is to take into account into the dynamic model the presence of integer approximation errors. In particular, we assume that a first error $\zeta(s)$ is introduced when $R(s)x(s)$ is replaced by its integer approximation, and a second error $\varrho(s)$ is due to the integer approximation of $u(s)$. The dynamic model now becomes

$$x(s+1) = R(s)x(s) + \zeta(s) + (u(s) + \varrho(s)), \tag{86}$$

in which $\|\zeta(s)\|_\infty \leq 0.5$, $\|\varrho(s)\|_\infty \leq 0.5$. We now review the problem constraints, considering the presence of these errors.

The constraint (76) should now be 'immunized' against approximation errors, i.e. it becomes

$$\mathbf{1}^T u(0) + \mathbf{1}^T \varrho(0) \leq C, \quad \forall \varrho(0) : \|\varrho(0)\|_\infty \leq 0.5,$$

which, since $\varrho(0) \in \mathbb{R}^m$, becomes

$$\mathbf{1}^T u(0) + \frac{m}{2} \leq C. \tag{87}$$

The conservation constraints (77) are equality constraints, and therefore impose a restriction on the allowable approximation errors $\varrho(s)$, $s = 1, \ldots, N-1$, which must hence be assumed to belong to the set $\Xi = \{z \in \mathbb{R}^m : \|z\|_\infty \leq 0.5,\ \mathbf{1}^T z = 0\}$. With this position, (77) remain unchanged.

For the team composition constraints (78), notice that setting

$$\varrho = [\varrho^T(0), \ldots, \varrho^T(N-1)]^T, \quad \zeta = [0_{1,m}, \zeta^T(1), \ldots, \zeta^T(N-1)]^T,$$

we have

$$x(s) = \Phi(s)\left(U + \varrho + \zeta\right),$$

and hence the constraints

$$x_L(s) \leq \Phi(s)\left(U + \varrho + \zeta\right) \leq x_U(s), \quad \forall \varrho \in \Xi^{(N)}, \zeta \in Z^{(N)}; \quad \text{for } s = 1, \ldots, N, \tag{88}$$

in which

$$\begin{aligned} \Xi^{(N)} &\doteq \{[z_1^T, z_2^T]^T : \|z_1\|_\infty \leq 0.5, z_2 \in \{\Xi \times \Xi \times \cdots \times \Xi\}, \\ Z^{(N)} &\doteq \{[0_{1,m}, z_1^T, \cdots, z_{N-1}]^T : z_i \in \mathbb{R}^m, \|z_i\|_\infty \leq 0.5, i = 1, \ldots, N-1\}. \end{aligned}$$

In turn, the constraints (88) are equivalent to

$$\begin{aligned} \Phi(s)U + \sup_{\varrho \in \Xi^{(N)}, \zeta \in Z^{(N)}} \Phi(s)(\varrho + \zeta) &\leq x_U(s), \\ \Phi(s)U + \inf_{\varrho \in \Xi^{(N)}, \zeta \in Z^{(N)}} \Phi(s)(\varrho + \zeta) &\geq x_L(s). \end{aligned}$$

The values of ϱ, ζ attaining the previous sup (say $\bar{\varrho}(s), \bar{\zeta}(s)$), and inf (say $\underline{\varrho}(s), \underline{\zeta}(s)$) are determined solving two linear programs, and therefore the composition constraints finally write

$$\begin{aligned} \Phi(s)(U + \bar{\varrho}(s) + \bar{\zeta}(s)) &\leq x_U(s), \quad (89) \\ \Phi(s)(U + \underline{\varrho}(s) + \underline{\zeta}(s)) &\geq x_L(s), \quad (90) \end{aligned}$$

for $s = 1, \dots, N$.

Finally, we notice that the constraints related to the objective can be treated similarly to the previous case. Specifically, the inequalities (83) now read

$$\begin{aligned} &\tilde{W}U + \sup_{\varrho \in \Xi^{(N)}, \zeta \in Z^{(N)}} \tilde{W}(\varrho + \zeta) \leq z, \\ &\tilde{W}U + \inf_{\varrho \in \Xi^{(N)}, \zeta \in Z^{(N)}} \tilde{W}(\varrho + \zeta) \geq -z, \\ &\sum_{i=1}^{Nm} z_i \leq \gamma, \end{aligned} \quad (91)$$

in which the values of ϱ, ζ attaining the extrema can again be computed solving two linear programs.

9.5 Resource Allocation under Uncertainty

The formulation introduced in the previous section hinges on the very unrealistic hypothesis that the values of the survival rates $r(s)$ at the various stages are exactly known. In the following, we relax this assumption and consider the problem of resource allocation under uncertainty. Specifically, we assume that the survival rate vectors $r(s)$ are of the form

$$r(s) = \bar{r}(s) + \delta(s), \quad s = 1, \dots, N-1,$$

in which $\bar{r}(s)$ is the known nominal value of the rate, and $\delta(s) \in \Delta(s)$ represent unknown 'fluctuations' or uncertainties around the nominal value, with $\Delta(s) \subseteq \mathbb{R}^m$ representing the allowable range of variation of the uncertainties.

A first idea in this respect would be to apply a 'robust optimization' methodology (see e.g. [25, 27]), and solve a version of problem (85) where the constraints are enforced for all admissible values of the uncertainty. This approach is however likely to be very conservative, since it neglects an important feature of the problem at hand, that is, there exist a stage schedule according to which the decisions have to be taken. To clarify the concept, we observe that not all the adjustments $u(s)$ need to be computed in advance (i.e. at the initial stage $s = 0$). Instead, only the decision $u(0)$ need to be taken at $s = 0$ (here-and-now decision), while before deciding for $u(1)$, we can wait and see what happens to the teams as they complete stage $s = 1$. In other words, the decision at $u(1)$ can benefit from the knowledge of the realization of the 'uncertainty' at $s = 1$. More generally, we observe that each decision $u(s)$, $s = 1, \dots, N-1$ can benefit from a 'basis of knowledge' of what happened from the initial stage up to s.

To exploit this information in a manageable way, we suppose that each decision vector $u(s)$ can be adjusted as a function of the realization of $r(s)$, and we explicitly set up an affine dependence of the form

$$u(s) = \bar{u}(s) + H(s)\delta(s), \quad (92)$$

in which $\bar{u}(s) \in \mathbb{R}^m$ and $H(s) \in \mathbb{R}^{m,m}$, $s = 0, \dots, N-1$ (with $H(0) \equiv 0_{m,m}$) are the new optimization variables. In more compact matrix form, we have that

$$U = \bar{U} + \bar{H}\bar{\delta},$$

in which $\bar{U} \doteq [\bar{u}^T(0) \cdots \bar{u}^T(N-1)]^T$ and $\bar{H} \doteq \mathbf{diag}\,(0_{m,m}, H(1), \dots, H(s-1))$ contain optimization variables, and $\bar{\delta} \doteq [0_{1,m}\ \delta^T(1)\ \cdots\ \delta^T(N-1)]^T \in \mathcal{D}$ contains the uncertainty terms, where $\mathcal{D} \doteq \{[0_{1,m}\ q] : q \in \Delta(1) \times \cdots \times \Delta(N-1)\}$.

We can now write the 'adjustable robust' version (see e.g. [24]) of our optimal allocation problem as

$$\min_{\gamma,z,\bar{U},\bar{H}} \gamma \quad \text{subject to:} \tag{93}$$

$$\mathbf{1}^T\bar{u}(0) \le C \tag{94}$$

$$\mathbf{1}^T\bar{u}(s) = 0; \quad s = 1, \dots, N-1 \tag{95}$$

$$\mathbf{1}^T H(s) = 0_{1,m}; \quad s = 1, \dots, N-1 \tag{96}$$

$$x_L(s) \le \Phi(s,\bar{\delta})\left(\bar{U} + \bar{H}\bar{\delta}\right) \le x_U(s), \tag{97}$$

$$\forall \bar{\delta} \in \mathcal{D}; \quad s = 1, \dots, N \tag{98}$$

$$-z \le \tilde{W}(\bar{U} + \bar{H}\bar{\delta}) \le z, \quad \forall \bar{\delta} \in \mathcal{D} \tag{99}$$

$$\sum_{i=1}^{Nm} z_i \le \gamma. \tag{100}$$

In this problem, we used the notation $\Phi(s,\bar{\delta})$ to underline the fact that the matrix $\Phi(s)$ defined in (80) depends on the survival rates $R(s) = \mathbf{diag}\,(r(s))$, $s = 1, \dots, N-1$, and hence on the uncertainty $\bar{\delta}$.

Problem (93)–(100) is a robust linear program, i.e. a linear program having a continuous infinity of constraints, see [24, 25]. In the mentioned papers, the authors show that in several 'tractable' cases the robust linear program can be converted *exactly* into a standard convex program having a finite number of constraints, and hence solved efficiently via interior point methods. Problem (93)–(100), however, does not fall among the tractable cases, since the uncertainty is affecting the problem data in a nonlinear way, and the 'recourse matrix' (i.e. the matrix $\Phi(s)$ that multiplies the adjustable variables, see [24]) is itself dependent on the uncertainty. Besides this technical difficulty, another motivation for not pursuing the worst-case approach is that this approach places equal importance on all possible uncertainty outcomes. In practical applications, one instead typically knows that some outcomes are 'more likely' than others, and may wish to exploit this knowledge when computing a solution.

We next describe a recently developed methodology for solving a probabilistic relaxation of problem (93)–(100).

9.6 Scenario-based optimization

The idea behind scenario-based solutions of robust linear programs is very simple: instead of considering the whole infinity of constraints of the problem, we consider only a finite number M of these constraints, selected at random according to a given probability distribution. Specifically, the constraints in (93)–(100) are parameterized by $\bar{\delta} \in \mathcal{D}$. Therefore, assuming a probability measure Π over $\mathcal{D}$, we first extract M (we shall discuss later 'how large' M

should be) independent and identically distributed samples of $\bar{\delta}$: $\bar{\delta}^{(1)}, \ldots, \bar{\delta}^{(M)}$, which constitute the uncertainty scenarios upon which we base our design. We remark that the choice of the probability measure Π now reflects our additional knowledge on which outcomes of the uncertainty are more likely than others. Subsequently, we solve the 'scenario counterpart' of the robust problem (93)–(100), which is defined below.

$$\min_{\gamma,\xi,\bar{U},\bar{H}} \gamma \quad \text{subject to:} \tag{101}$$

$$\mathbf{1}^T \bar{u}(0) \leq C \tag{102}$$

$$\mathbf{1}^T \bar{u}(s) = 0; \quad s = 1, \ldots, N-1 \tag{103}$$

$$\mathbf{1}^T H(s) = 0_{1,m}; \quad s = 1, \ldots, N-1 \tag{104}$$

$$x_L(s) \leq \Phi(s, \bar{\delta}^{(i)}) \left(\bar{U} + \bar{H}\bar{\delta}^{(i)} \right) \leq x_U(s), \tag{105}$$

$$i = 1, \ldots, M; \quad s = 1, \ldots, N \tag{106}$$

$$-\xi \leq \tilde{W}(\bar{U} + \bar{H}\bar{\delta}^{(i)}) \leq \xi, \quad i = 1, \ldots, M \tag{107}$$

$$\sum_{i=1}^{Nm} \xi_i \leq \gamma. \tag{108}$$

A first immediate consideration about (101)–(108) is that it is a standard linear program (with a possibly large, but finite number of constraints), which is easily solvable by LP numerical codes. A fundamental question is however related to what kind of guarantees of robustness can be provided by a solution that a priori satisfies only a finite number M of selected constraints. The good news in this respect is that, if we sample a sufficiently large number of constraints, then the scenario solution will be 'approximately feasible' for the robust problem (93)–(100), i.e. the probability measure of the set of uncertainties such that the corresponding constraints are violated by the scenario solution goes to zero rapidly as M increases. This result has been recently derived in [26], and it is next contextualized to the problem at hand.

9.7 Approximate feasibility of scenario solutions

Consider a generic robust LP in the form

$$\min_x c^T x \text{ subject to } A(\xi)x \leq b, \quad \forall \xi \in \mathcal{X}, \tag{109}$$

wherein $x \in \mathbb{R}^n$ and $\mathcal{X} \subseteq \mathbb{R}^\ell$ is a closed set, and no restrictions are imposed on the dependence of the data matrix A on ξ. Assume that (109) is feasible, and suppose that a probability measure Π is imposed on $\mathcal{X}$. Then, the scenario counterpart of (109) is the linear program

$$\min_x c^T x \text{ subject to } A(\xi^{(i)})x \leq b, \quad i = 1, \ldots, M, \tag{110}$$

in which $\xi^{(i)}$, $i = 1 \ldots, M$ are iid samples of $\xi \in \mathcal{X}$, extracted according to probability Π. Assume further that (110) has a unique optimal solution x^* (this uniqueness assumption is technical and could be removed, see [26]). Clearly, the scenario solution x^* depends on the random sample $\xi^{(i)}$, $i = 1 \ldots, M$, and it is therefore itself a random variable. The following theorem highlights the 'approximate feasibility' property of this solution.

Theorem 1. *Fix a probabilistic risk level $\epsilon \in (0,1)$ and a confidence level $\beta \in (0,1)$, and let x^* be the optimal solution of the scenario problem (110), computed with*

$$M \geq \frac{n}{\epsilon\beta} - 1. \tag{111}$$

Then, with probability at least $1-\beta$,

$$\text{Prob}\{\xi \in \mathcal{X} : A(\xi)x^* \not\leq b\} \leq \epsilon. \tag{112}$$

In other words, the probability of the set of uncertainties that violate the inequality $A(\xi)x^* \leq b$ can be made arbitrarily small by sampling a sufficient number of scenarios, and therefore we say that the scenario solution is (with high probability $1-\beta$) approximately feasible for the robust problem (109), i.e. it satisfies all but a small set of the original constraints.

9.8 A posteriori analysis

It is worth noticing that a distinction should be made between the a priori and a posteriori assessments that one can make regarding the probability of constraint violation for the scenario solution. Indeed, *before* running the optimization, it is guaranteed by Theorem 1 that if $M \geq n/\epsilon\beta - 1$ samples are considered, the solution of the scenario problem will be ϵ-approximately feasible, with probability no smaller than $1-\beta$. However, the a priori parameters ϵ, β are generally chosen not too small, due to technological limitations on the number of constraints that one specific optimization software can deal with.

On the other hand, once a scenario solution has been computed (and hence $x = x^*$ is fixed), one can make an a posteriori assessment of the level of feasibility using Monte-Carlo techniques. In this case, a new batch of $\tilde{M}$ independent random samples of $\xi \in \mathcal{X}$ is generated, and the *empirical probability* of constraint violation, say $\hat{V}_{\tilde{M}}(x^*)$, is computed according to the formula $\hat{V}_{\tilde{M}}(x^*) = \frac{1}{\tilde{M}} \sum_{i=1}^{\tilde{M}} 1(A(\xi^{(i)})x^* \leq b)$, where $1(\cdot)$ is the indicator function. If $V(x^*) \doteq \text{Prob}\{\xi \in \mathcal{X} : A(\xi)x^* \not\leq b\}$ denotes the true violation probability, the classical Hoeffding's inequality [28] states that

$$\text{Prob}\{|\hat{V}_{\tilde{M}}(x^*) - V(x^*)| \leq \tilde{\epsilon}\} \geq 1 - 2\exp(-2\tilde{\epsilon}^2\tilde{M}),$$

from which it follows that $|\hat{V}_{\tilde{M}}(x^*) - V(x^*)| \leq \tilde{\epsilon}$ holds with confidence greater than $1-\tilde{\beta}$, provided that

$$\tilde{N} \geq \frac{\log 2/\tilde{\beta}}{2\tilde{\epsilon}^2} \tag{113}$$

test samples are drawn. This latter a posteriori test can be easily performed using a very large sample size $\tilde{N}$ because no optimization problem is involved in such an evaluation.

Returning to our resource allocation problem, the solution procedure that we propose is the following one.

1. Select the a priori probabilistic risk level ϵ and confidence β, and compute the number of necessary scenarios according to (111). We remark that experimental numerical experience showed that the actual probabilistic levels achieved by the scenario solution are usually *much* better than the ones established by means of Theorem 1. This fact suggest in practice not to insist on too small a priori levels.

2. Solve the scenario LP (101)–(108), obtaining the optimal variables $\gamma^*, z^*, \bar{U}^*, \bar{H}^*$.

3. Test a posteriori the obtained solution via Monte-Carlo, using a large sample size, to determine a very reliable estimate of the actual probability of violation of the scenario solution. If this level of probability is acceptable for the application at hand, we are finished, otherwise we may try another scenario design step, taking into account a larger set of sampled scenarios, and iterate the procedure.

9.9 Interaction models

In this section, we propose an iterative heuristic for the solution of a modified allocation problem. Consider the generic robust LP problem formulated in (109). In deriving the scenario counterpart of this problem, we assumed that a *fixed* probability distribution was assigned on the uncertain parameter ξ. In terms of the actual resource allocation problem, this assumption means that the survival rates are random variables, and that we know a priori their probability distributions. However, a more realistic model of the problem should be able to take into account *interaction* effects between the decision variables and the uncertainties. By interaction we here mean that the probability distribution of the survival rates of a certain stage could be itself dependent on the amount of resources that we commit for that stage. For instance, the overall chance of surviving a given stage may increase if we send more resources to that stage.

A way of modeling this interaction in our generic framework (109) is to assume that the probability distribution on $\xi \in \mathcal{X}$ depends on x, that is, we assign a *conditional* distribution $\Pi(\xi|x)$ on ξ. Clearly, if interaction is present, we can no longer directly apply the scenario approach, since the correct distribution according to which we need to sample the scenarios is unknown. We therefore propose the following iterative heuristic to solve the problem in presence of interaction.

1. Let an initial solution $x^{(k)}$, $k = 0$ be available;

2. Draw random scenarios $\xi^{(1)}, \ldots, \xi^{(M)}$ according to probability $\Pi(\xi|x^{(k)})$, and solve the resulting scenario problem. Let $k \leftarrow k + 1$, and denote by $x^{(k)}$ the optimal scenario solution;

3. Repeat 2., until some suitable convergence condition is reached.

The effectiveness of this heuristic needs to be tested on numerical examples.

9.10 Numerical examples

In this section, we address the problem of allocating UAVs (Unmanned Aerial Vehicles) optimally and dynamically in order to perform various sequential tasks where risk is present due to hostile opponents. In practice, it is often required to allocate UAVs in teams in order to perform various sequential tasks in a hostile environment in which their survival rates are uncertain. This makes this particular application a good illustration of our method.

9.11 The nominal problem

In this problem, we consider that our opponent has 5 different types of equipment, namely small SAM (surface to air missile), medium SAM, large SAM and the Early Warning Radars (EWRs). All kinds of SAMs have destructive capability. However, the EWRs and the Long SAM-fcs(fire control sensor) work as tracking and sensing tools and don't have any destructive capability. Their destructive ranges are given in Table 1. The enemy equipment with higher destructive ranges are riskier to destroy than that with lower destructive ranges.

	Small SAM	Medium SAM	Large SAM	Long Sam-fcs	EWR
Range (km)	25	50	100	0	0

Table 1: Ranges of opponent's equipments

The controller needs to assign UAVs to teams in order to perform six main tasks, which are destroying 6 enemy EWRs, namely EWR1, EWR2, EWR3, EWR4, EWR5 and EWR6. However, due to the presence of other enemy SAMs, it is not possible to destroy all the 6 targets with an acceptable risk level, unless some other enemy SAMs are destroyed. As a result, in order to perform the main tasks under an acceptable risk level, we need to destroy other targets first. We define primary targets as the targets which are originally assigned to be destroyed and secondary targets as the targets that need to be destroyed in order to reduce the risk inherent to the mission to the primary targets. Hence, the assignment problem becomes a sequential and a dynamic one. We perform the assignment in 'waves' (stages): we start the assignment process by forming teams in the first wave in a way that they destroy some assigned targets that are under a threshold risk level. Once the targets are destroyed at the first stage, the risk for the targets to be destroyed in the second wave is reduced under the threshold level. At the end of the first wave, we reassign the team composition among the survived UAVs in order to destroy the targets assigned for the second wave. We keep reassigning till we destroy all the assigned targets in all the stages. In this experiment, the targets to be destroyed in various stages are input data, as described in Table 2.

According to the input data, we define $m_p = [7\ 6\ 4\ 6]^T \in \mathbb{R}^N$, where N denotes the total number of stages, and $m_p(s)$ denotes the number of tasks at the stage s. In order to be consistent with notations described earlier, we define $m := \max_i(m_p(i))$ and therefore we assign $m = 7$ tasks at each stage. However, if the number of tasks is $l < m$ in stage s, we add extra slack $(m - l)$ tasks in that stage, such that $0 \leq x_i(s) \leq 0$ for all $l < i \leq m$. We also assume that the cost for UAV allocation at the first wave is 1 unit and the transaction cost in the later waves is TC units, where TC is an experimental variable. The total number of available UAV is 40. The nominal survival rate of an UAV assigned to destroy medium SAM, long SAM, and EWRs are 0.65, 0.5, and 1 respectively. Moreover, an additional constraint is imposed: at any stage, at least one UAV is required to be assigned to each target.

We ran the experiment with different transition costs $TC = 0, 0.9, 1, 100, 000$ and the resulting team constitutions are as shown in Table 3-6. Later, when we mention $TC = \infty$, we actually mean that the experimental run was performed with $tc = 100, 000$. Total cost and the total number of UAVs required for the various assignments are summarized in Table 7.

It is clear from Table 7 that the total cost and the total number of UAVs required increase with the increase of transition cost. When $TC = \infty$, the team assignment becomes static and produces higher total cost.

If the survival rates are certain and accurate, the assignments obtained by using our algorithms produce the minimum

Threat Name	Objective Classification	wave
Medium SAM 27	Secondary	1
EWR3	Primary	1
EWR 1	Primary	1
Long SAM-fcs 3	Secondary	1
Medium SAM 5	Secondary	1
Medium SAM 3	Secondary	1
Medium SAM 28	Secondary	1
Long SAM 14	Secondary	2
EWR 2	Primary	2
Long SAM 2	Secondary	2
Medium SAM 9	Secondary	2
Medium SAM 2	Secondary	2
Medium SAM 30	Secondary	2
Long SAM-fcs 4	Secondary	2
Medium SAM 10	Secondary	3
Long SAM 8	Secondary	3
EWR 4	Primary	3
Medium SAM 12	Secondary	3
Long SAM 5	Secondary	4
Medium SAM 13	Secondary	4
Long SAM 7	Secondary	4
Medium SAM 14	Secondary	4
EWR 5	Primary	4
EWR 6	Primary	4

Table 2: Tasks

cost, while using the minimum number of UAVs.

9.12 The Robust counterpart

In the experiment, we assume that the survival rates of UAVs while encountering SAMs are not certain. Instead, the survival rates are stochastic. We suppose for the purpose of this example that the survival rates while encountering medium SAMs and long SAMs follow uniform distributions $\mathbf{U}[0.45\ , 0.55]$ and $\mathbf{U}[0.6\ , 0.7]$ respectively. Using the algorithm discussed in Section 9.6, we ran the experiment with variable TC. We randomly picked 25 sample points by using these distributions in order to obtain the constraints. We ran each of the experiment 20 times. Although the algorithm does not always guarantee a team assignment that satisfies all the constraints, it does it in most of the tested cases. We ran each of the experiment 20 times. In each successful run with a fixed TC, our algorithms produce assignments that yield a total cost and the total number of UAVs required to complete all the tasks. We compute the averages of these two quantities over all the successful runs. The summary is shown in Table 8.

We observe that even with the presence of uncertainty in the survival rate, our algorithm performs well. In most of the runs, the stochastic algorithm is able to satisfy all the constraints.

	Stage 1	Stage 2	Stage 3	Stage 4
Team 1	Medium SAM27 (1)	Long SAM14 (1)	Medium SAM10 (1)	Long SAM5(1)
Team 2	EWR3 (3)	EWR2 (4)	Long SAM8 (1)	Medium SAM13(1)
Team 3	EWR1 (3)	Long SAM2 (1)	EWR4(4)	Long SAM7(1)
Team 4	Long SAM-fcs3 (1)	Medium SAM9(1)	Medium SAM12 (1)	Medium SAM14(1)
Team 5	Medium SAM5 (1)	Medium SAM30(1)	(0)	EWR5(1)
Team 6	Medium SAM3 (1)	Long SAM-fcs4(1)	(0)	EWR6(1)
Team 7	Medium SAM29 (1)	(0)	(0)	(0)

Table 3: Task Assignment with $TC = 0$

	Stage 1	Stage 2	Stage 3	Stage 4
Team 1	Medium SAM27 (1)	Long SAM14 (1)	Medium SAM10 (1)	Long SAM5(1)
Team 2	EWR3 (5)	EWR2 (5)	Long SAM8 (1)	Medium SAM13(1)
Team 3	EWR1 (2)	Long SAM2 (1)	EWR4(4)	Long SAM7(1)
Team 4	Long SAM-fcs3 (1)	Medium SAM9(1)	Medium SAM12 (1)	Medium SAM14(1)
Team 5	Medium SAM5 (1)	Medium SAM30(1)	(0)	EWR5(1)
Team 6	Medium SAM3 (1)	Long SAM-fcs4(1)	(0)	EWR6(1)
Team 7	Medium SAM29 (1)	(0)	(0)	(0)

Table 4: Task Assignment with $TC = 0.2$

Moreover, we ran an experiment where the survival rate is stochastic but the nominal team assignments are used. We record the percentage of time the nominal assignment produces successful run (satisfies all the constraints) and compare the results with the robust counterpart. The comparison is shown in Figure 29. We observe that robust algorithm provides successful assignment significantly more often than the nominal counterpart under uncertainty.

We conclude that the robust assignment algorithm based on robust linear program performs very well even if the survival rate is uncertain.

9.13 Conclusion

We have proposed a strategic planning scheme that allocates resources to groups of tasks organized in successive stages. Our algorithms allocate the resources to the tasks (i.e. form 'teams') by dynamically re-organizing the teams at each stage, while minimizing a cost objective over the whole stages horizon. Furthermore, we have proposed an algorithm based on 'linear programming with adjustable variables,' that can solve uncertain linear program by means of the sampled scenarios randomized technique. We have applied our algorithm to a problem of UAVs allocation in an uncertain and risky environment. We have shown that our model provides an optimal solution to the problem while satisfying all the constraints in most of the runs.

In the specific context of UAV allocation, many further issues remain open for numerical investigation. First, we have here considered a fixed statistical model for the survival rates. However, a model that takes into account 'interactions' (see Section 9.9) or at least a saturation on the survival rates seems better suited for the application at hand. Also, we would like to add origin-destination dependent transaction costs at each stage in our model, as

	Stage 1	Stage 2	Stage 3	Stage 4
Team 1	Medium SAM27 (2)	Long SAM14 (1)	Medium SAM10 (1)	Long SAM5(1)
Team 2	EWR3 (4)	EWR2 (4)	Long SAM8 (4)	Medium SAM13(1)
Team 3	EWR1 (5)	Long SAM2 (5)	EWR4(3)	Long SAM7(1)
Team 4	Long SAM-fcs3 (1)	Medium SAM9(1)	Medium SAM12 (1)	Medium SAM14(1)
Team 5	Medium SAM5 (2)	Medium SAM30(1)	(0)	EWR5(1)
Team 6	Medium SAM3 (2)	Long SAM-fcs4(1)	(0)	EWR6(1)
Team 7	Medium SAM29 (1)	(0)	(0)	(0)

Table 5: Task Assignment with $TC = 1$

	Stage 1	Stage 2	Stage 3	Stage 4
Team 1	Medium SAM27 (8)	Long SAM14 (5)	Medium SAM10 (2)	Long SAM5(1)
Team 2	EWR3 (2)	EWR2 (2)	Long SAM8 (2)	Medium SAM13(1)
Team 3	EWR1 (2)	Long SAM2 (2)	EWR4(2)	Long SAM7(1)
Team 4	Long SAM-fcs3 (8)	Medium SAM9(4)	Medium SAM12 (2)	Medium SAM14(1)
Team 5	Medium SAM5 (4)	Medium SAM30(2)	(1)	EWR5(1)
Team 6	Medium SAM3 (4)	Long SAM-fcs4(2)	(1)	EWR6(1)
Team 7	Medium SAM29 (2)	(1)	(1)	(1)

Table 6: Task Assignment with $TC = \infty$

discussed in Remark 1, as well as different types of UAVs.

	Total Cost	Total Number of UAVs
$TC = 0$	11	11
$TC = 0.2$	15.02	13
$TC = 1$	23.98	17
$TC = \infty$	30	30

Table 7: Total cost incurred and total number of UAVs required

	Average Total Cost	Average Total Number of UAVs	% Successful run
$TC = 0$	12.06	12	85%
$TC = 0.2$	15.56	13	90%
$TC = 1$	26.20	17	80%
$TC = \infty$	37	37	85%

Table 8: Total cost incurred and total number of UAVs required

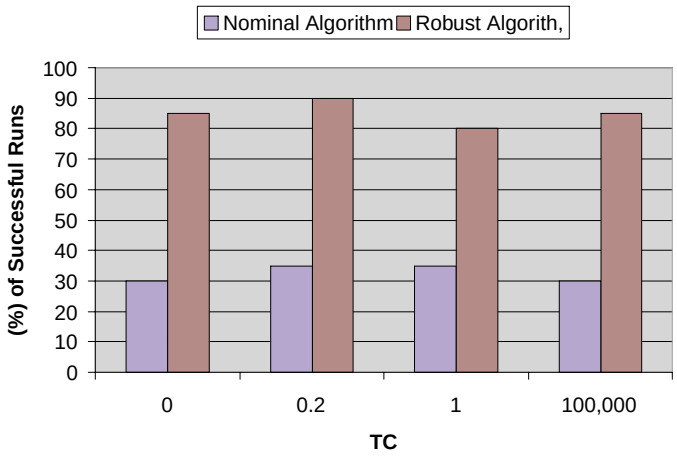


Figure 29: Percentage of the runs when all the constraints are satisfied.

10 Module 8: Path planning with multiple constraints

The fast marching algorithm is a very efficient technique for finding least cost paths, in the absence of any constraint. This module extends the use of this algorithm when there are constraints.[19]

10.1 Introduction

Few problems are as well studied as the path planning or routing problem; it appears in engineering disciplines that vary from robotics to wireless communication to matrix factorization. A major challenge in developing solutions to the problem are the many, sometime subtle, variations it can adopt: the topology of the state space and cost metrics, the types of acceptable paths, the number of sources and destinations, the acceptable degree of optimality, etc. While every variant has at least one solution method—enumerate all feasible paths until an acceptably optimal one is found—the key to developing efficient solution algorithms is to take advantage of the particular properties of the variant of interest.

In this section we examine the path planning problem in a continuous state space subject to constraints on additive path integral cost metrics. The original motivation for this work was the planning of fuel constrained flight paths for unmanned aerial vehicles through enviroments with varying levels of threat. Paths are generated by gradient descent on a value function (with no local minima), which is the solution of an Eikonal partial differential equation (PDE).[20] Path integral costs are evaluated by solving an auxiliary PDE. Both PDEs can be solved quickly for low dimensional systems, thus yielding a practical algorithm for path planning. Because both PDEs are solved over the entire state space, paths to any possible destination can be rapidly evaluated.

To handle constraints, we sample the Pareto optimal surface looking for paths with feasible combinations of costs. The sampling method only reaches the convex hull of the Pareto surface, so for nonconvex problems it may not always find the optimal feasible path; however, in our experience the degree of nonconvexity has not been enough to cause significant problems.

The asymptotic cost of the algorithm is $\mathcal{O}(MdN^d \log N)$, where M is the number of sampled points on the Pareto optimal surface, d is the state space dimension, and N is the number of grid points in each state space dimension. To adequately sample the Pareto surface, M will typically be exponential in the number of separate cost functions k. While these two exponentials are daunting, in practice the algorithms described below are quite practical on the desktop when the sum $k + d$ is less than around five or six; for example, section 10.3 includes a problem in two dimensions with three cost functions that is solved in less than one minute on the authors' laptop computer.

Gradient descent on a value function solution of the Eikonal equation has been used previously for unconstrained, single cost path planning problems. The innovative contribution of this section is the application of auxiliary PDEs to calculate multiple path integral costs, and the use of those costs to find constrained optimal paths.

In the remainder of this section we formally outline our path planning problem and examine related work. Subse-

[19]The research reported here was conducted by Ian M. Mitchell and Shankar Sastry and supported by ONR under MURI contract N00014-02-1-0720. A condensed version of this section appears in the CDC 2003 proceedings.

[20]Classical applications of the Eikonal PDE are in the fields of optics and seismology. Its solution can be interpreted as a first arrival time or a cost to go, depending on whether the boundary conditions represent sources or sinks.

quent sections describe the algorithm, provide several examples, and discuss extensions to more general problems.

We work in a state space $\mathbb{R}^d$. Unless otherwise specified, norms are Euclidean $\|\cdot\| = \|\cdot\|_2$. Let $\mathbb{R}^+ = (0, \infty)$ be the set of strictly positive real numbers.

10.1.1 Problem definition

A *path* $p : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d$ is parameterized by an arclength $s \in \mathbb{R}^+$ and a destination location $x \in \mathbb{R}^d$. Assume that all paths have a single *source location* $x_s \in \mathbb{R}^d$ (we will relax this assumption later). The *path cost functions* $\{c_i(x)\}_{i=1}^k$, where $c_i : \mathbb{R}^d \to \mathbb{R}^+$, are continuous, bounded and strictly positive. The cost along a path is additive, so the total cost of a path can be evaluated by a *path integral*

$$P_i(x) = \int_0^T c_i(p(s,x))ds, \qquad \text{where } \begin{cases} p(0,x) & = x_s, \\ p(T,x) & = x. \end{cases} \tag{114}$$

In words, $P_i : \mathbb{R}^d \to \mathbb{R}$ is the total cost, according to path cost function $c_i(\cdot)$, of following the path $p(\cdot, x)$ from the source location x_s to the point x.

As an example, consider planning the flight path of an aircraft from its base at x_s to various destinations. The most obvious path cost function is fuel, which we approximate as a constant $c_{\text{fuel}}(x) = c_{\text{fuel}}$. A second path cost function might be the threat of inclement weather $c_{\text{weather}}(x)$. A third might be uncertainty about the enviroment, encoded as $c_{\text{uncertain}}(x)$. The latter two costs are inhomogenous, meaning that their value depends on x. Examples of cost functions are shown in figures 31 and 34.

There are two related problems that we might wish to solve starting from the parameters x_s and $\{c_i(x)\}_{i=1}^k$ described above. Given some set of *cost constraints* $\{C_i\}_{i=1}^k$, where $C_i \in \mathbb{R}^+$, we might want to find feasible paths such that $P_i(x) \leq C_i$ for all $i = 1 \ldots k$. Alternatively, we might try to minimize $P_1(x)$ subject to constraints on the remaining costs $P_i(x) \leq C_i$ for all $i = 2 \ldots k$. In either case, we will usually be interested in quantitative measures of the tradeoffs between the various path cost functions; for example, in the second type of problem what relaxation of the constraint C_2 would be required to cut the cost $P_1(x)$ in half?

10.1.2 Related work

The significance of the most closely related algorithmic work [2, 3, 4] is discussed in section 10.2.4. However, similar problems have been investigated in several other fields.

Path planning is a central endeavor in robotics research [5], so we mention only the most closely related work. The algorithm discussed in this section could be categorized as a potential field approach [6], in the sense that the paths are determined by gradient descent on a scalar function defined over the state space. In particular, the value function constructed in section 10.2 is an example of a navigation function [7]—a potential field free of the local minima that hinder most potential field methods (although in general it will contain saddle points). The specific use of the Eikonal equation for robot path planning in the single cost case was examined in [8], and is equivalent to the approach used in [9].

Independently, the networking community has been solving constrained shortest path planning on discrete graphs [10, 11, 12], primarily for the purpose of network routing. While this research involves problems with multiple costs, it makes the assumption that the number of distinct cost values possible at any node in the graph is finite and bounded. The resulting algorithms are pseudo-polynomial time: polynomial in the size of the graph and in the value of any constraints. If we seek a convergent approximation for the continuous path planning problem, we cannot assume that the value function can be discretized and thus we cannot use pseudo-polynomial time algorithms. It should be noted, however, that our method for exploring the Pareto optimal surface of possible path costs by sampling values of λ (see section 10.2.3) is equivalent to the fastest algorithm proposed for finding constrained shortest paths in [12]. The distinction between their algorithm and ours is the underlying shortest path problem: discrete in their case, continuous in ours.

The related work that is closest mathematically is a tomographic application [13], which uses the Eikonal equation (115) to calculate travel time and a version of the path integral PDE (117) to determine perturbations of a linearized form of the Eikonal equation. To our knowledge, the use of (117) for evaluating path integral costs is original.

10.2 Value function solution

We discuss the value function method for finding the shortest path in the single cost case, and then how to compute path integrals along value function generated paths. With these tools we can explore the range of paths that might meet the constraints when multiple cost functions are involved. This section concludes with a discussion of an efficient algorithm for solving the required differential equations.

10.2.1 Single objective shortest path

Consider the simplest case $k = 1$ with a single path cost function $c(x) = c_1(x)$ (because it will be used to generate a value function, we call this cost $c(x)$ the *value cost function*). It can be shown that the minimum cost to go from the source x_s to any point x in the state space is the solution of the inhomogenous Eikonal equation

$$\begin{aligned} \|\nabla V(x)\| &= c(x) \quad \text{for } x \in \mathbb{R}^d, \\ &\text{with boundary condition } V(x_s) = 0. \end{aligned} \tag{115}$$

The solution $V : \mathbb{R}^d \to \mathbb{R}$ of this PDE is called the *value function*. In practice V is rarely differentiable and therefore (115) does not have a solution in the classical sense. The viscosity solution [14] is the appropriate weak solution for the shortest path problem. In section 10.2.4 we shall discuss algorithms for computing accurate numerical approximations of the viscosity solution of (115), but for now the important fact is that efficient schemes exist for problems of reasonably low dimension.

Given the viscosity solution V, the optimal path $p^*(\cdot, x)$ can be determined by gradient descent of V from a fixed target location x. In practical terms, let $\hat{p}(s, x)$ be a path that starts at a particular x and terminates at x_s. Then $\hat{p}$ is

the solution to the ordinary differential equation (ODE)

$$\frac{d\hat{p}(s,x)}{ds} = \frac{\nabla V(\hat{p}(s,x))}{\|\nabla V(\hat{p}(s,x))\|} \quad \text{for } s \in \mathbb{R}^+ \text{ and fixed } x \in \mathbb{R}^d, \\ \text{with initial condition } \hat{p}(0,x) = x. \tag{116}$$

We stop extending the solution at some $\hat{s}$ such that $\hat{p}(\hat{s},x) = x_s$. Then $\hat{s} = T$ is the arclength of the shortest path from x_s to x, and that path is given by $p^*(s,x) = \hat{p}(T-s,x)$. Because $V(x)$ is the cost to get to x from x_s along path p^*, the path integral for this path is $P^*(x) = V(x)$. The gradient descent (116) cannot get stuck in local minima because V has none.[21] In theory, (116) can get stuck at saddle points of V, but the stable manifolds of such points are of measure zero in the state space, and are thus unlikely to be a problem in practical implementations subject to floating point roundoff noise.

10.2.2 Computing path integrals

Throughout the remainder of this section, we consider only paths generated by (116) for some value function V. In this section we examine how to compute the path integral when the value cost function is not the same as the path cost function. To differentiate the two cost functions, we denote the value cost function in (115) by $c(x)$ and the path cost function in (114) by $c_i(x)$. Both must use the same source location x_s.

Starting from the differential form of (114), we formally derive a PDE for the path integral $P_i(x)$

$$\begin{aligned} \frac{dP_i(p(s,x))}{ds} &= c_i(p(s,x)), \\ \frac{\partial P_i(p(s,x))}{\partial p(s,x)} \cdot \frac{dp(s,x)}{ds} &= c_i(p(s,x)), \\ \nabla P_i(p(s,x)) \cdot \frac{\nabla V(p(s,x))}{\|\nabla V(p(s,x))\|} &= c_i(p(s,x)), \\ \nabla P_i(p(s,x)) \cdot \nabla V(p(s,x)) &= c_i(p(s,x))c(p(s,x)), \end{aligned}$$

where (116) is used in the second step and (115) is used in the third. Consequently, for all reachable points in the state space,

$$\nabla P_i(x) \cdot \nabla V(x) = c_i(x)c(x) \quad \text{for } x \in \mathbb{R}^d, \\ \text{with boundary condition } P_i(x_s) = 0. \tag{117}$$

Because the cost structure is isotropic (independent of path direction) the system is small time controllable and for our single source version all states will be reachable. The derivation above assumes that all the functions involved are differentiable, but as was stated earlier this assumption will fail for $V(x)$ and therefore likely also for $P_i(x)$. We are in the process of developing a robust proof that the viscosity solution of (117) is the path cost integral we seek.

When solving (117), $P_i(x)$ is the unknown while $V(x)$, $c_i(x)$ and $c(x)$ are all known. Not surprisingly, (115) can be recovered from (117) for the single cost case of the previous section by substituting $c_i(x) = c(x)$ and $P_i(x) = V(x)$.

[21] Easily seen if V is differentiable, since a local minimum would require $\nabla V(x) = 0$, but $c(x) > 0$. A more rigorous argument based on the positivity of c can be constructed when V is a viscosity solution.

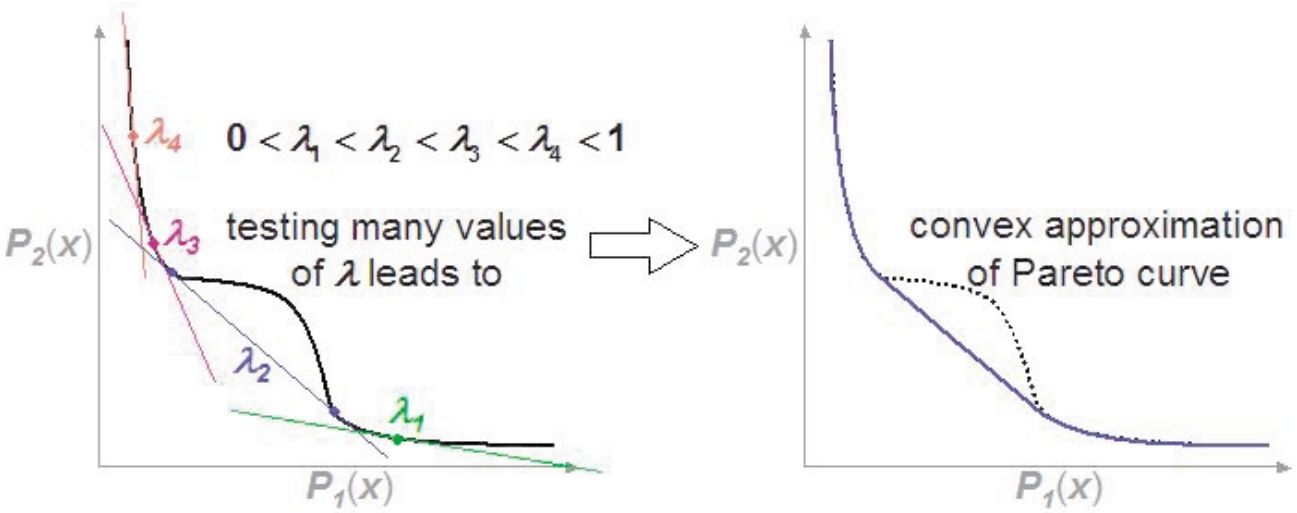


Figure 30: Pareto optimal curve for a particular destination state x. Left: each value of λ samples a point on the curve. Right: testing all values of λ would yield a convex approximation of the Pareto curve.

10.2.3 Exploring potential paths

As discussed in section 10.1.1, one of our goals was an algorithm to generate feasible paths subject to a collection of cost constraints. In the previous two sections we described PDEs whose solutions were a path generating value function V in (115) and the path integrals P_i for those paths in (117). The remaining missing ingredient is the value cost function $c(x)$ in (115). In this section we discuss the results of using convex combinations of the path cost functions as the value cost function.

We start with the simplest multiobjective case, $k = 2$. Let

$$c^\lambda(x) = \lambda c_1(x) + (1 - \lambda)c_2(x) \quad \text{for some } \lambda \in [0, 1].$$

Then evaluate (115) and (117) for $V^\lambda(x)$, $P_1^\lambda(x)$ and $P_2^\lambda(x)$. The first thing to notice is that $\lambda = 1$ calculates paths optimal in c_1 and $\lambda = 0$ paths optimal in c_2. Therefore, if $P_1^{\lambda=1}(x) > C_1$ or $P_2^{\lambda=0}(x) > C_2$ for some point x, there cannot be any feasible paths from x_s to x. Intermediate values of λ will generate paths lying somewhere between these two extremes.

Testing all possible values of λ would effectively construct the convex hull of the Pareto optimal tradeoff curve between the two cost functions. Figure 30 shows a possible Pareto curve for a single point x, the points on that curve determined by several values of λ, and the convex hull of that curve. A point on the curve is a pair $(P_1^\lambda(x), P_2^\lambda(x))$ and lies where a line of slope $\frac{\lambda}{(\lambda-1)}$ is tangent to the Pareto curve. Therefore, λ is a quantitative measure of the tradeoff between the two cost functions.

In general the Pareto curve is not convex, so this method may fail to detect a feasible path even if one exists. Nonconvexity in the Pareto curve will manifest itself by jumps in the values of the path integrals $P_1^\lambda(x)$ and $P_2^\lambda(x)$

for fixed x as λ is varied continuously; for example, consider the jump in path integrals as λ is varied in the range $[\lambda_2 - \epsilon, \lambda_2 + \epsilon]$ for some small $\epsilon > 0$ in figure 30. However, neighboring values of λ can be used to bound the error in the convex approximation and nonconvexity has not been a problem in our experience. It should be pointed out that the Pareto curve characterised above is for a single point x in the state space. Because V^λ and P_i^λ are calculated over the entire state space, the technique actually approximates a separate Pareto curve for all points x.

To handle the case $k > 2$, we simply choose a set $\{\lambda_j\}_{j=1}^k$ such that $\lambda_j \in [0, 1]$ for all j and $\sum_{j=1}^k \lambda_j = 1$. Then $c^{\{\lambda_j\}}(x) = \sum_{j=1}^k \lambda_j c_j(x)$, and we can solve for the corresponding value function and path cost integrals. In this case it is the convex hull of the Pareto optimal surface that is explored as the set $\{\lambda_j\}$ is varied.

10.2.4 Numerical algorithms

The discussion above would be nothing more than a mathematical diversion if it were not possible to solve (115), (116) and (117) numerically for some practical problems. In this section we briefly outline an existing efficient algorithm for solving (115), and modify that algorithm slightly to handle (117) as well. We postpone implementation details to section 10.3.4.

To treat (116), we assume that (115) and (117) can be computed for a variety of λ values to generate $V^\lambda(x)$ and $\{P_i^\lambda(x)\}_{i=1}^k$. Then a particular $\hat{\lambda}$ is chosen such that any path integral constraints are satisfied ($P_i^{\hat{\lambda}}(x) \leq C_i$). A path is determined by solving (116) for value function $V^{\hat{\lambda}}(x)$ with a standard ODE integration method, such as Runge-Kutta.

Solving (115) efficiently relies on an algorithm first described in [2], although the explanation that follows is based on an independently developed equivalent version [3] commonly known as the *Fast Marching Method* (FMM). This algorithm is basically the Dijkstra algorithm for computing shortest paths in a discrete graph [15], suitably modified to deal with a continuous state space. For readers interested in alternatives, there are other algorithms for solving (115); for example, [16, 17].

The value function $V(x)$ is approximated on a Cartesian grid over the state space with N nodes in each dimension, for a total of N^d nodes. Direct application of Dijkstra's algorithm on this discrete Cartesian graph remains a popular approximation method for this problem; however, the paths generated by such an approximation measure their cost metrics in a coordinate dependent manner,[22] and are visibly segmented at the grid's resolution. In contrast, FMM approximations can generate paths with subgrid resolution (see section 10.3.1); paths that are reasonably smooth for practically sized grids. Furthermore, these approximations are theoretically convergent, meaning that the approximation approaches the true value function solution of (115) as $N \to \infty$ on all of the state space except a subset of measure zero.

We initialize the FMM by setting $V(x_s) = 0$ and $V(x_m) = \infty$ for all other nodes x_m (in practice we choose a large floating point value for ∞). We also place x_s into a list ℓ. At each iteration of the FMM we remove the node x_m in ℓ with minimum value $V(x_m)$; this value is now fixed. We update $V(x_n)$ for each neighbor x_n of x_m with

[22] For example, Dijkstra on a square Cartesian grid measures distance with the Manhattan or 1-norm; in this norm the distance between two points depends on the alignment of the coordinate axes. While this axis alignment bias can be reduced by adding more edges to the graph, it will persist unless every possible path is enumerated by making the graph completely dense. The solution of the Eikonal equation (115) measures distance in the Euclidean or 2-norm, which is independent of axis alignment.

$V(x_n) > V(x_m)$. If any of those neighbors were not in ℓ already, we place them in ℓ. We then repeat, taking the node of next smallest value from ℓ and updating its neighbors, until no more nodes remain in ℓ. This procedure is the basis of Dijkstra's algorithm. Each node is removed from ℓ only once and has a constant number of neighbors. As we will see, updating each neighbor takes a constant amount of time. If a heap [18] or something similar is used to sort ℓ, the smallest node can be determined in logarithmic time, for a total cost $\mathcal{O}(dN^d \log N)$.

The only difference between Dijkstra's method and an FMM lies in the update equation for a node x_n [3]. Instead of considering each neighbor of x_n separately, we form a first order upwind finite difference approximation of $\nabla V(x_n)$ using various combinations of the neighbors x_p whose values $V(x_p)$ are less than the current approximation of $V(x_n)$. Plugging these finite difference approximations into (115) yields an implicit quadratic equation for the new approximation of $V(x_n)$. Detailed update formulas are given in the appendix.

To solve (117), we use an approximation scheme outlined in [4]. The "extension velocity" $F_{\text{ext}}(x)$ described in that paper is computed by solving

$$\nabla F_{\text{ext}}(x) \cdot \nabla V(x) = 0,$$

which is just (117) with a zero right hand side. In practice, we integrate the computation of $P_i(x)$ into the FMM computation of $V(x)$. When each node x_m is removed from ℓ, we compute $P_i(x_m)$ for each i. The first order upwind finite difference approximation of $\nabla V(x_m)$ is already known from the last update of $V(x_m)$, while $c_i(x_m)$ and $c(x_m)$ can be directly evaluated. Forming a first order upwind finite difference approximation of $\nabla P_i(x_m)$ using the same neighbor nodes that were used to build the approximation of $\nabla V(x_m)$ yields an implicit linear equation for $P_i(x_m)$. Note that the neighbors x_p of x_m involved with the approximation of $\nabla V(x_m)$ will all have $V(x_p) < V(x_m)$, so they will all have been removed from ℓ before x_m and hence will have known values $P_i(x_p)$. Again, detailed update formulas are given in the appendix.

10.3 Examples

For our example we consider planning a path for an aircraft flying across the idealized unit square country from lower left to upper right. The first cost function will be fuel, which we assume is a constant $c_{\text{fuel}}(x) = c_{\text{fuel}} = 1$. Because these are toy examples, we provide no specific units for our cost functions.

The second cost function will represent the threat of weather related problems $c_{\text{weather}}(x)$. Note that the intuitive quantification of weather threat would be the probability of encountering a storm along the flight path. This quantification cannot be used as a cost because probabilities are not additive; however, under an independence assumption they can be transformed into an additive cost by a logarithmic transformation. The figures and tables below assume that this transformation has been performed in generating $c_{\text{weather}}(x)$ from meteorologically determined storm probabilities.

Ideally, this weather forecast would be an accurate short term estimate of weather threat. When we examine a three cost example in section 10.3.2, we will assume that part of our fictional country is well monitored and can thus generate accurate short term weather threat estimates, while another part of the country is poorly monitored and in this region we are forced to resort to long term climatological estimates. Because these long term estimates are less accurate, we introduce a third cost function $c_{\text{uncertain}}(x)$ which will penalize paths through the poorly instrumented region of the country.

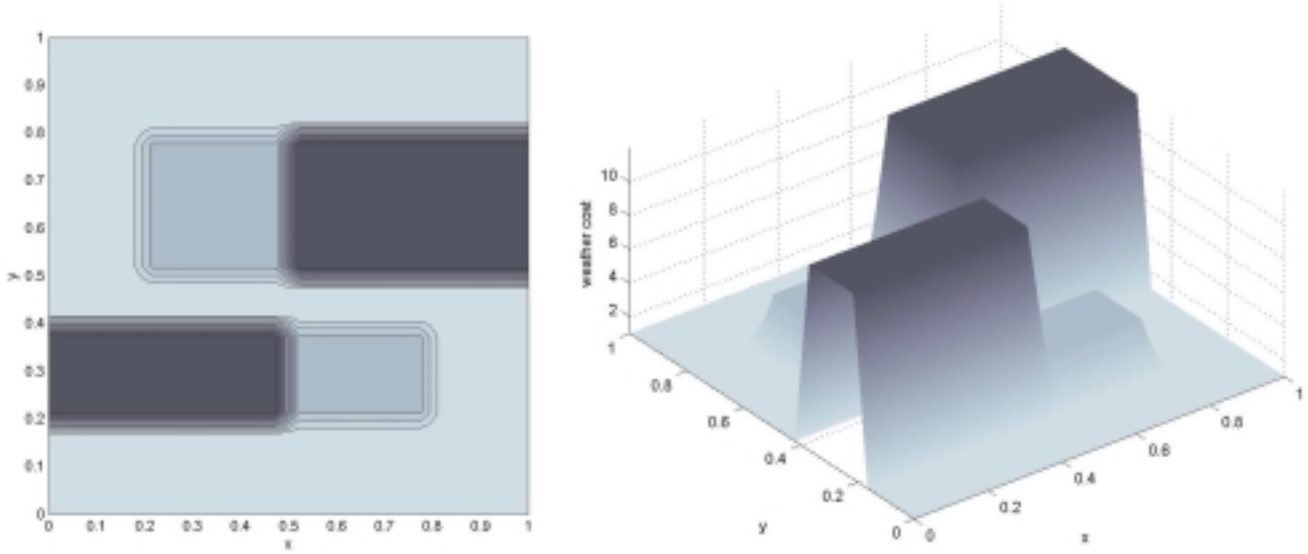


Figure 31: Weather threat cost function $c_{\text{weather}}(x)$

We focus on two dimensional examples primarily because three dimensional paths are very challenging to visualize on paper. While three dimensions is noticeably more expensive, we demonstrate in section 10.3.4 that it can still be done at interactive rates on the desktop.

The gradient descent procedure that generates the paths (explained in section 10.3.4) produces a series of waypoints leading from the source to the destination. In the plots that follow there is a small gap between the source location and the start of the paths. This gap appears because the source location is not explicitly added to the waypoint list; the gap is choosen small enough that the aircraft can fly a direct line between the source and the first waypoint (the beginning of the plotted path).

10.3.1 Two costs in two dimensions

Figure 31 shows a simple weather threat cost map $c_{\text{weather}}(x)$. Notice that the lower high threat bar extending from the left is slightly thinner than the upper high threat bar extending from the right.

Figure 32 shows four example paths plotted for various combinations of $c_{\text{weather}}(x)$ and c_{fuel} from the source $x_s = \begin{bmatrix}0.1 & 0.1\end{bmatrix}^T$ (marked by a star symbol) to the destination $x_d = \begin{bmatrix}0.9 & 0.9\end{bmatrix}^T$ (marked by a plus symbol). The combinations are described in table 9. In searching for paths that satisfy the fuel constraints, the range of λ was sampled uniformly. The λ values shown for the two constrained paths are the largest sampled λ for which the fuel constraint was satisfied. Notice in particular that the path under tight fuel constraints (the solid line) prefers to cross the thinner lower bar of the weather cost function rather than the fuel symmetric path that exists crossing the thicker upper bar.

line type	minimize what cost?	fuel constraint	fuel cost	weather cost	λ
dotted	fuel	none	1.14	8.81	0.00
solid	weather	1.3	1.27	4.55	0.13
dashed	weather	1.6	1.58	3.03	0.62
dash dot	weather	none	2.69	2.71	1.00

Table 9: Properties of paths in figure 32.

line type	minimize what cost?	fuel constraint	weather constraint	fuel cost	weather cost	uncertainty cost
dotted	fuel	none	none	1.14	8.83	1.52
dash dot	weather	none	none	2.74	2.75	5.96
dashed	uncertainty	none	none	1.18	8.42	1.19
solid	uncertainty	1.3	6.0	1.25	5.85	1.25

Table 10: Properties of paths in figure 35.

Figure 33 shows the points on the Pareto optimal curve of the destination location x_d generated by a uniform sampling of the space $\lambda \in [0, 1]$. Two explanations exist for those regions where the sample points are well spaced—either the uniform sampling was too coarse, or the Pareto curve is nonconvex. In the former case, a more intelligent sampling strategy could fill in the gaps inexpensively. Furthermore, even if the curve is nonconvex the existing samples provide fairly tight bounds on the degree of possible nonconvexity.

10.3.2 Three costs in two dimensions

The first two cost functions are the same as in section 10.3.1: constant fuel $c_{\text{fuel}} = 1$ and $c_{\text{weather}}(x)$ from figure 31. For the third cost function, we assume that the upper left corner of our mythical country has few weather stations and therefore we create the uncertainty cost function $c_{\text{uncertain}}(x)$ shown in figure 34.

The resulting paths from x_s to x_d are shown in figure 35 and described in table 10. Because they optimize the same costs in the same manner, the dotted and dash dotted paths are basically the same as those shown in figure 32.[23] The most interesting path is that denoted by the solid line. Notice that the constraints on this path were satisfied by the tight fuel constrained path (also a solid line) in figure 32. In this case, however, we are penalizing paths that travel in the upper left portion of the map with the uncertainty cost $c_{\text{uncertain}}(x)$. Therefore, a path that crosses the thick high cost portion of the upper bar of the weather cost map is chosen; even though the weather cost is higher, it is still within the specified constraint and the resulting path's uncertainty cost is nearly as good as the minimum uncertainty cost path (given by the dashed line).

[23] The slight differences between their tabulated costs in tables 9 and 10 is due to the coarser state space grid used in this three constraint example.

line type	minimize what cost?	fuel constraint	fuel cost	weather cost
dotted	fuel	none	1.41	3.54
dashed	weather	none	1.64	1.64
solid	weather	1.55	1.55	2.00

Table 11: Properties of paths in figure 36.

10.3.3 Two costs in three dimensions

For a three dimensional problem, we plot a path from $x_s = \begin{bmatrix}0.1 & 0.1 & 0.1\end{bmatrix}^T$ to $x_d = \begin{bmatrix}0.9 & 0.9 & 0.9\end{bmatrix}^T$. The fuel cost function c_{fuel} remains a constant, while the weather cost $c_{\text{weather}}(x)$ has five stormy regions centered at the points:

$$\begin{bmatrix}0.1\\0.9\\0.9\end{bmatrix} \quad \begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix} \quad \begin{bmatrix}0.9\\0.1\\0.1\end{bmatrix} \quad \begin{bmatrix}0.9\\0.5\\0.5\end{bmatrix} \quad \begin{bmatrix}0.1\\0.5\\0.5\end{bmatrix}$$

Each stormy region adds a scaled and shifted gaussian to $c_{\text{weather}}(x)$. In order to represent the general low level threat of unforeseen weather disturbances, we set $c_{\text{weather}}(x) = 1$ anywhere that the sum of the storm costs drops below unity. In order to break the symmetry of the problem, the first stormy region is 50% larger than the remaining four. A visualization of the weather cost function is shown in figure 36 along with three paths from x_s to x_d. The three volumetric shells in the figure represent (from faintest to darkest) the 1.1, 2.0 and 3.0 isosurfaces of $c_{\text{weather}}(x)$; its peak value is 5.5. The three demonstration paths are described in table 11.

10.3.4 The implementation and deExecution times

To compute approximations of $V(x)$ and $P_i(x)$, we have implemented a version of the FMM described in section 10.2.4 in C++ for Cartesian grids. While the code itself can handle any dimension, in practice the physical memory limits of desktop machines restrict the dimension to at most five even with very coarse grids. Using a MEX interface, these PDE solving routines can be called directly from Matlab.

To handle cost constrained paths, the sampling over λ is performed in Matlab. The speed of Matlab's interpreted language is not an issue in this case, because the inner loop of the λ sampling process is the compiled C++ but still relatively time consuming FMM algorithm. In all of the examples shown, the range of λ is sampled uniformly. If a particular destination point were known in advance, a directed sampling of λ could quickly yield more accurate results; for example, bisection in the two cost case. In fact, if a particular destination point is specified, the FMM can be run faster in some cases by applying a version of A* search on the list ℓ, rather than just selecting the node with minimum value ([11] discusses this technique in the discrete graph setting).

We have so far been generating a relatively small number of paths, so this process is handled with Matlab's extensive ODE integration facilities. Once a $\hat{\lambda}$ has been chosen such that any path constraints are satisfied, $V^{\hat{\lambda}}(x)$ is used in (116) to determine the path. Because $\nabla V^{\hat{\lambda}}$ may change direction significantly from one integration step to

section	d	k	N	$\Delta\lambda$	time (min)
10.3.1	2	2	201	0.005	0.5
10.3.2	2	3	101	0.020	1.0
10.3.3	3	2	101	0.010	22.3

Table 12: Run parameters for the examples (d = dimension, k = number of constraints, N = grid size, $\Delta\lambda$ = sample interval of convex combination cost). Time includes generation of the cost functions, PDE and ODE solves, and plotting all the figures.

N	time (s) per λ	ratio
51	0.01	
101	0.04	3.24
201	0.13	3.76
401	0.55	4.20
801	2.44	4.41

Table 13: Costs of refining the grid in two dimensions. Time per λ is the time to solve a single instance of the PDEs for $V(x)$ and two path integral costs $P_1(x)$ and $P_2(x)$. The ratio column shows the roughly quadratic growth in execution time as N is increased.

the next, (116) is a moderately stiff ODE. Consequently, we have found a variable stepsize, implicit trapezoidal integrator to be effective (Matlab's `ode23t`); however, other variable stepsize integrators—such as the high order explicit 4-5 Runge-Kutta—could also be used.

We summarize the parameters for the examples of the previous sections in table 12. The grid sizes and number of λ samples were chosen large enough to give decent results and not so large as to overburden the authors' desktop computer. These timings and those below are for a 2 GHz, 1 GB Pentium 4 Dell Inspiron 8200 running Windows XP Professional, Matlab version 6.5 (release 13) and Matlab's lcc compiler.

Tables 13 and 14 demonstrate the costs of refining the PDE grid. Both tables assume only two path integral cost PDEs are solved; however, most of the time in the FMM algorithm is spent solving for $V(x)$, so the increase in time per λ sample of an additional path integral PDE is only 10%–20%. As mentioned previously, the asymptotic cost of the algorithm is $\mathcal{O}(dN^d \log N)$ per λ sample. The ratio columns of the two tables show the expected growth in execution time—slightly above quadratic with N for two dimension, slightly above cubic with N for three—in all but the coarsest two dimensional grids (where the roughly constant overhead of initialization will be relatively significant).

The time to solve (116) to generate a particular path is largely independent of the dimension or grid size, and is completely independent of the number of constraints or λ sampling interval. It will depend on the destination location (the closer to the source, the shorter the path). In our experience, most paths can be generated in less than a second, and few take more than three seconds. Using a compiled integrator (rather than Matlab's interpreted

N	time (s) per λ	ratio
51	1.27	
101	12.66	9.99
201	125.46	9.91

Table 14: Costs of refining the grid in three dimensions. Time per λ is the time to solve a single instance of the PDEs for $V(x)$ and two path integral costs $P_1(x)$ and $P_2(x)$. The ratio column shows the slightly greater than cubic growth in execution time as N is increased.

routines) would speed this process up even further.

To demonstrate the effects of refining or coarsening the grid on the quality of the resulting paths, figure 37 shows the paths from the example in section 10.3.1 generated for three different grid resolutions. The paths shown in figure 32 correspond to the $N = 201$ case. While grid refinement does yield visibly better paths, even the coarsest grid gets a qualitatively correct answer on even the most convoluted path.

10.4 Discussion

We have demonstrated an algorithm for constrained path planning in continuous state spaces for additive cost metrics and isotropic but inhomogenous and nonconvex cost functions. In those cases with multiple cost functions, a convex approximation of the Pareto optimal surface is explored; consequently, the algorithm may not find all feasible paths although in practice this has rarely been a problem. While the asymptotic cost of the algorithm is exponential in the dimension and in the number of cost functions (assuming uniform sampling of the Pareto optimal surface), it can be run at interactive rates on the desktop if their sum is five or less, and overnight if their sum is six.

There are several straightforward extensions of this work to more general path planning problems. We can immediately incorporate multiple source locations, by making each source a boundary condition with value zero of the PDEs (115) and (117). The resulting value function will generate paths from the nearest source to each destination state. Hard obstacles in the state space can be treated by either making the cost function very large in their interior or by making the obstacle's boundary a part of the PDEs' boundaries with very large value. Creating boundary nodes with intermediate values (neither zero nor very large) can be interpreted as penalizing those nodes as possible source locations. We can also swap the meaning of source and destination, in which case the value function can be used to generate a feedback control.

The basic FMM algorithm described in section 10.2.4 has been extended to unstructured meshes, and a more accurate second order approximation scheme has been developed. For more details on FMM and its extensions, we refer the reader to [19]. We are in the process of developing a version of FMM that runs on an adaptively refined Cartesian grid, so as to better represent problems with hard obstacles.

The current path planning formulation assumes that the cost of a path is a function only of its current state; this is equivalent to claiming that the vehicle which will execute the path can travel equally well in any direction from any

location. This assumption is reasonable when the resolution of the grid is much coarser than the dynamics of the vehicle; for example, planning aircraft paths across a country. But on shorter time and space scales, it is unrealistic to assume that an airplane can make a sharp turn. Treating nontrivial vehicle dynamics requires that the cost functions be anisotropic. Unfortunately, such anisotropy means that the value function will no longer be the viscosity solution of the eikonal equation (115), but rather a more general static Hamilton-Jacobi PDE. The FMM will not work on these PDEs; however, several algorithms have been proposed to solve them quickly [16, 20, 21, 22].

The additive path integral cost model used in this section is very common, and includes multiplicative costs through a logarithmic transformation. Another common cost metric is maximum cost along the path. While maximum cost can currently be evaluated for a single path during the integration of (116), we are investigating methods capable of evaluating this metric over the entire state space. We are also examining efficient methods of approximating all of the Pareto optimal curve for each destination location, rather than just its convex hull.

Acknowledgements: We would like to thank Aniruddha Pant for suggesting the sweeping procedure over convex combinations of the multiple cost functions, and Professor Pravin Varaiya for the interpretation of this procedure as a convex approximation of the Pareto optimal surface and for several very useful discussions of value function properties. Thanks are also due to the Berkeley MICA team for providing the examples which originally motivated this work.

10.5 Appendix: Update equations for any number of dimensions

Section 10.2.4 described the basic FMM algorithm used to solve (115) for $V(x)$ and (117) for $P_i(x)$. In this appendix we give the update equations that form the heart of these algorithms: first for $V(x)$ and then for $P_i(x)$. These update equations are independent of dimension d, but work only on Cartesian grids. The update algorithm and equation for $V(x)$ given below is a version of those given in the appendix of [8], modified to treat grids with dimensionally dependent spacing (where h_j is the grid spacing in dimension j).

When a node x_m is removed from the list ℓ, any neighbor x_n with $V(x_n) > V(x_m)$ may need to be updated. Consider a specific neighbor node, which we label x_0. This node will have 2^d neighbors itself: one in each direction (we will call these directions left and right) in each dimension. Choose a set of neighbor indices I by picking the neighbor (either left or right) with lowest value from each dimension (so I has d elements). If the grid spacing is equal in all dimensions, the nodes x_j for $j \in I$ and the node x_0 are the vertices of a d dimensional simplex; otherwise they form a distorted simplex. The formula derived below calculates $V(x_0)$ as if the characteristic of (115) giving $V(x_0)$ its value came from this simplex.

It is possible that the characteristic in question flows along a lower dimensional face of the simplex rather than through its interior. Now we identify the subset of indices J from which the characteristic arises. First we define the

following terms, where all of the summations are over the index set J (excepting the indices explicitly excluded).

$$T_1 = \sum_j \left(\sum_{l \neq j} h_l^2 \right) V(x_j),$$

$$T_2 = \sum_j \left(\sum_{l \neq j} h_l^2 \right) c^2(x_0),$$

$$T_3 = \sum_{j_1} \sum_{j_2 \neq j_1} \left(\sum_{l \neq j_1, j_2} h_l^2 \right) \left[V(x_{j_1}) - V(x_{j_2}) \right]^2,$$

$$T_4 = \sum_j \sum_{l \neq j} h_l^2.$$

To find the appropriate J, start with $J = I$. While

$$T_2 < T_3, \tag{118}$$

keep removing the node x_j with largest value $V(x_j)$ in J. Once $T_2 \geq T_3$, use the remaining nodes in J to form first order upwind finite difference approximations of the partial derivatives of V at x_0, and plug these approximations into the square of (115) to get

$$\sum_{j \in J} \left(\frac{V(x_j) - V(x_0)}{h_j} \right)^2 = c^2(x_0).$$

We can then use the quadratic equation to solve for $V(x_0)$.

$$\hat{V}(x_0) = \frac{T_1 + \left(\sum_j h_j \right) \sqrt{T_2 - T_3}}{T_4}. \tag{119}$$

The reader can verify that condition (118) ensures that the resulting discriminant in (119) is positive. If the resulting value $\hat{V}(x_0)$ is less than the existing value $V(x_0)$, then $\hat{V}(x_0)$ is taken as the new approximation of V at x_0.

Now consider the update of $P_i(x_m)$. Let $x_0 = x_m$ and remember the set J last used to update $V(x_0)$. Form first order upwind finite difference approximations for the partial derivatives of P_i and V at x_0, and plug these approximations into (117) to get

$$\sum_{j \in J} \left(\frac{P_i(x_j) - P_i(x_0)}{h_j} \right) \left(\frac{V(x_j) - V(x_0)}{h_j} \right) = c_i(x_0) c(x_0).$$

Rearranging the terms yields the update equation (the sums are again over J)

$$P_i(x_0) = \frac{\left(\sum_j \left[\sum_{l \neq j} h_l^2 \right] P_i(x_j) \left[V(x_j) - V(x_0) \right] \right) - c_i(x_0) c(x_0) \sum_j h_j^2}{\left(\sum_j \left[\sum_{l \neq j} h_l^2 \right] \left[V(x_j) - V(x_0) \right] \right)}.$$

Because this equation is solved only once for each x_0 and each P_i when that node x_0 is removed from list ℓ, computing the path integral cost functions is much cheaper per cost function than computing the value function.

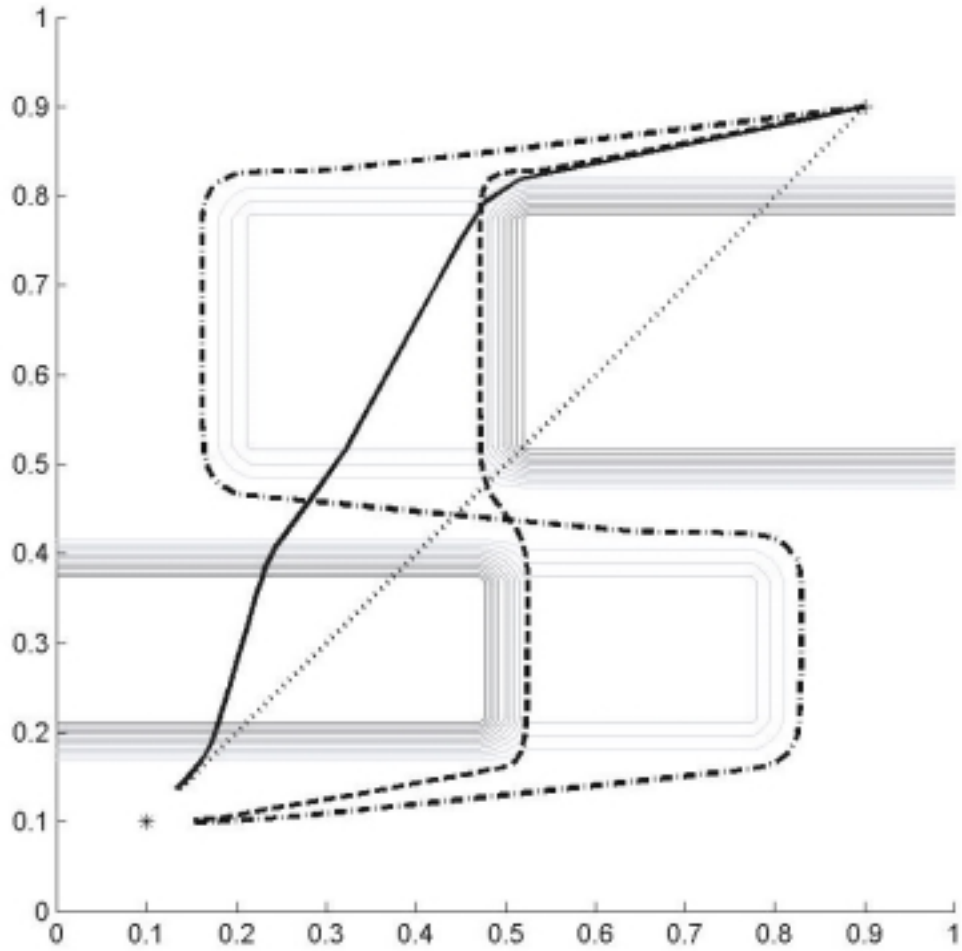


Figure 32: Some fuel and weather constrained paths. The properties of each path are explained in table 9.

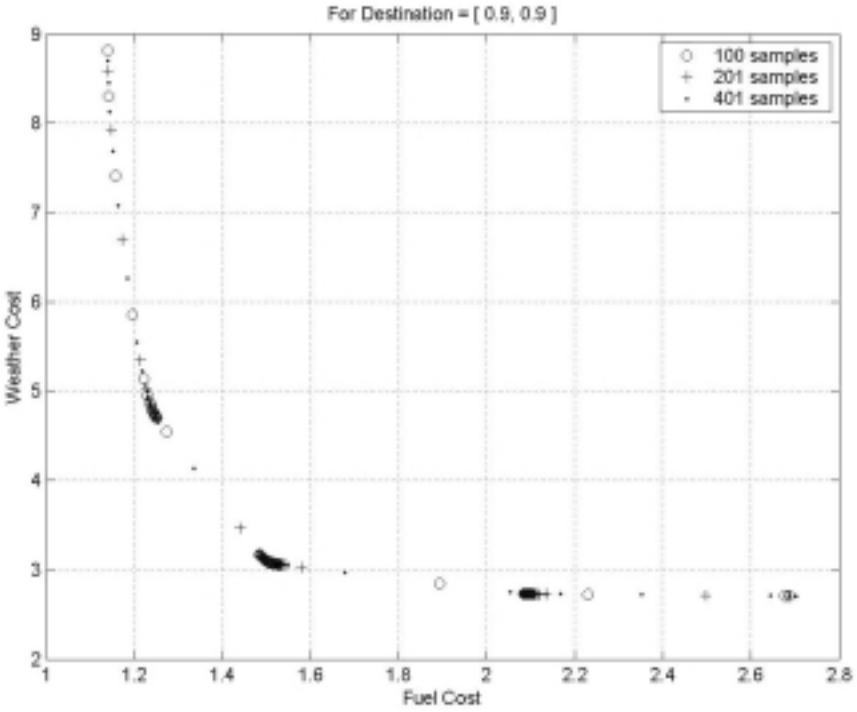


Figure 33: Sampling the Pareto optimal curve for a particular destination point.

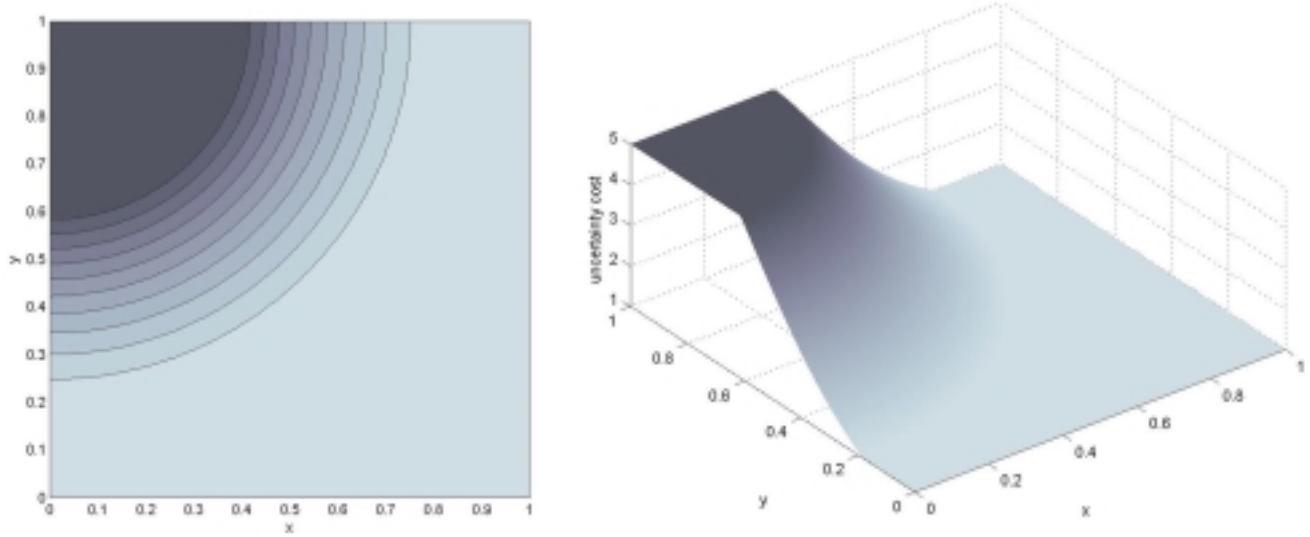


Figure 34: Uncertainty cost function $c_{\text{uncertain}}(x)$

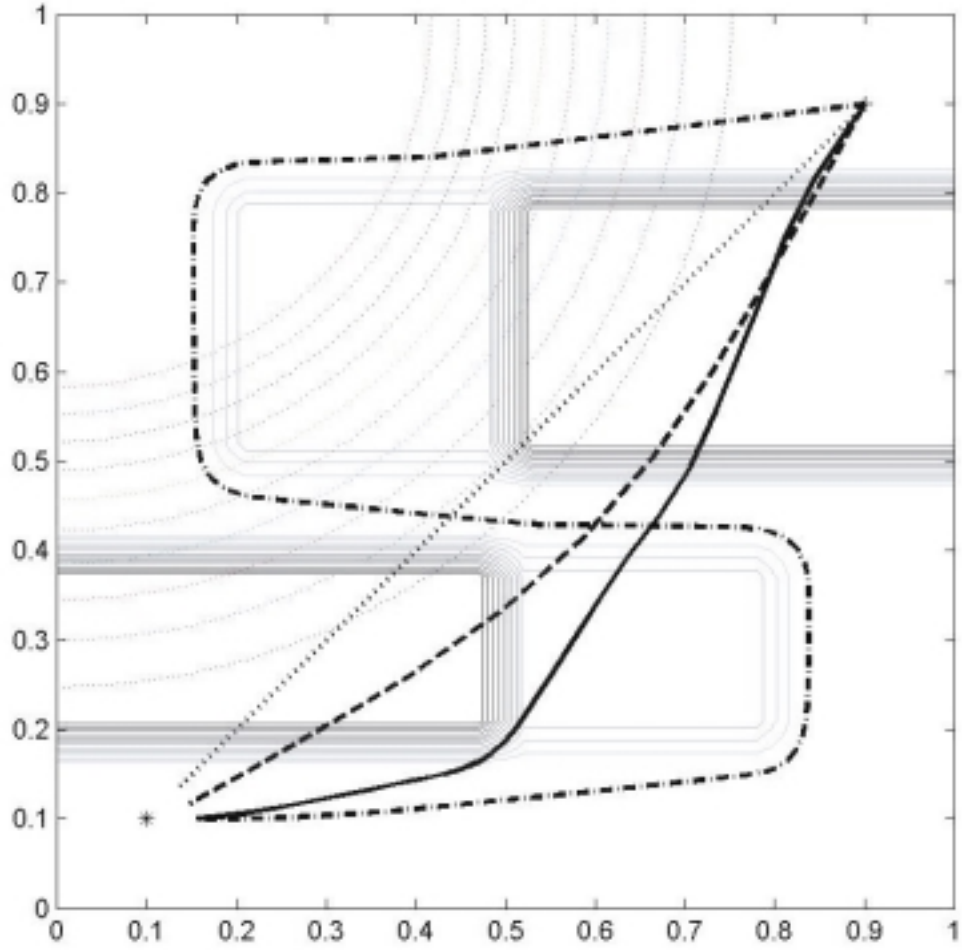


Figure 35: Some fuel, weather and uncertainty constrained paths. The properties of each path are explained in table 10.

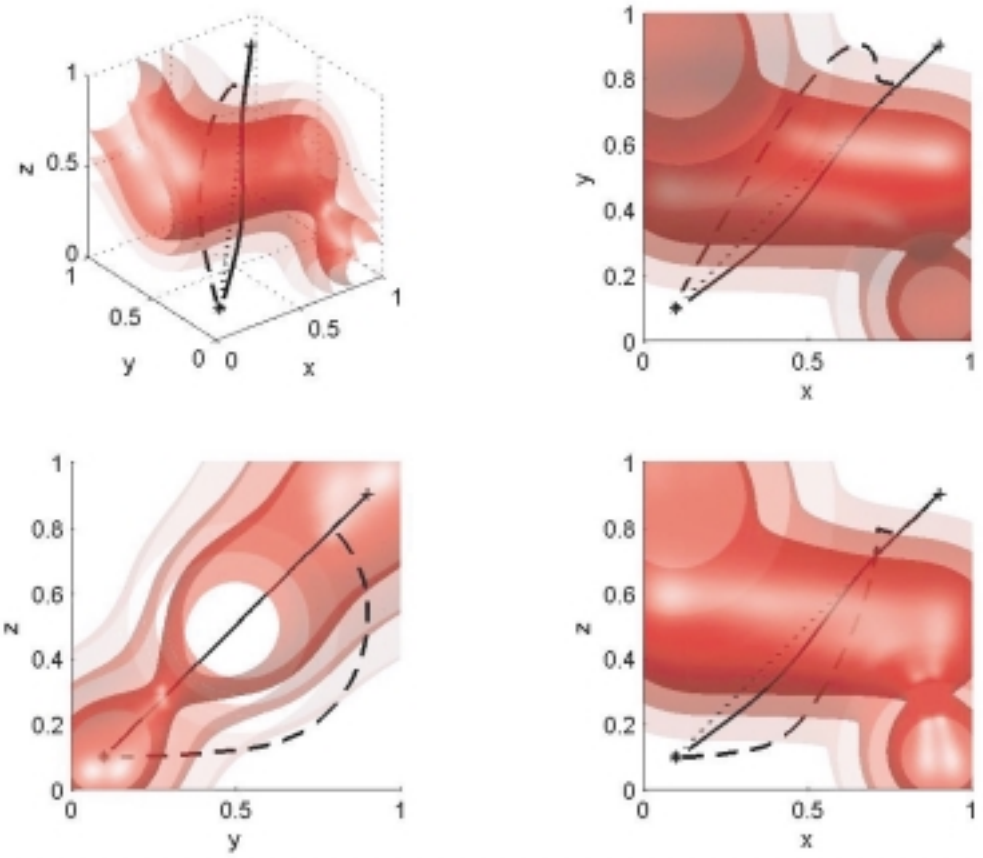


Figure 36: Some fuel and weather constrained paths in three dimensions. The properties of each path are explained in table 11.

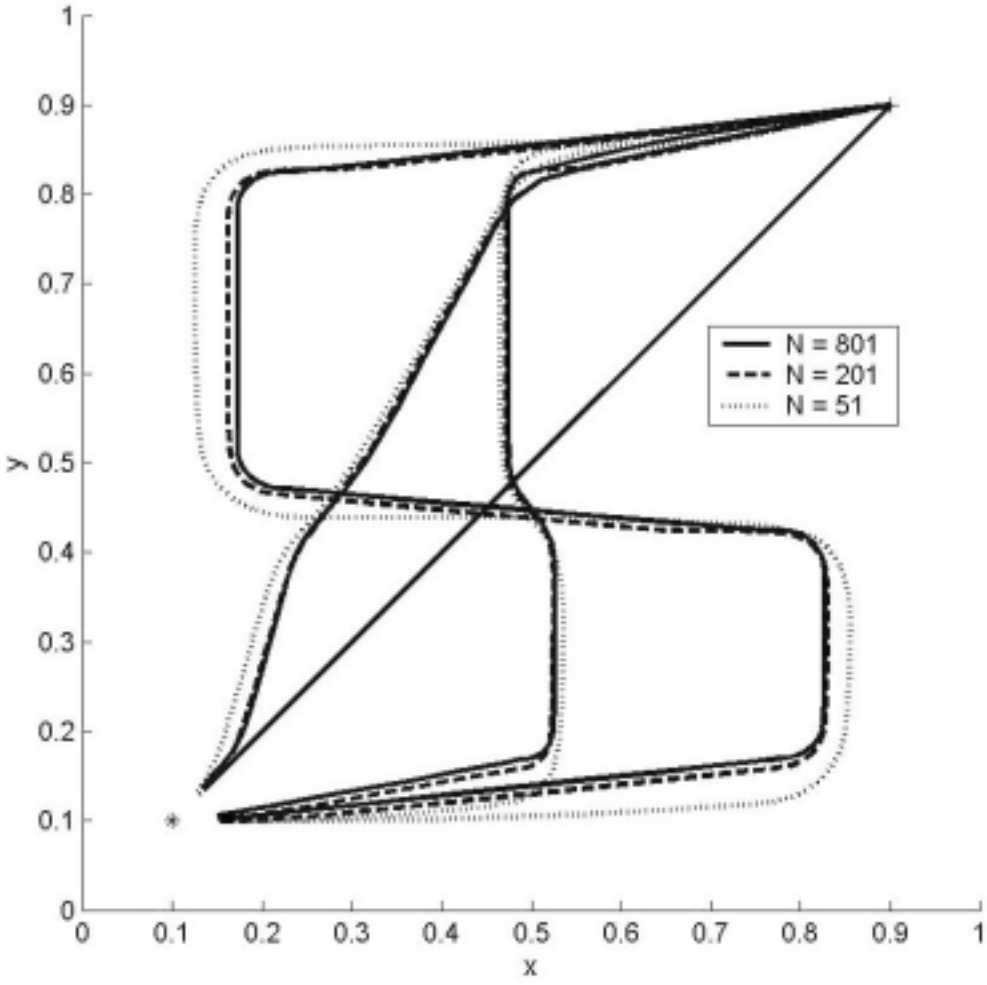


Figure 37: Comparing the path approximations generated by various grid resolutions. As the grid resolution improves, for almost every destination point the approximation converges to the analytically optimal path.

11 Conclusions

We summarize our major contributions relative to the MICA program objectives, which are to

- Develop theory, algorithms, software, and modeling/simulation capabilities for hierarchical battlespace management and distributed control of semi-autonomous entities
  - Cultivate dynamic operational and mission planning for teamed entities
  - Develop cooperative path/execution planning
  - Address an active, intelligent adversary and threats in an uncertain environment
- Demonstrate multiple vehicle execution of team-based strategies

Following the guidance of the program management, battlespace management comprises the two phases of 'offline planning' and 'online execution'. We organize the latter in a two-layer hierarchy of 'team coordination' and 'UAV control', figure 1. To deal with additional information that might arrive during the execution, we introduce a 'state estimation' procedure, described in section 6, which can trigger 're-planning', figure 2. Our work can be situated within this structure and compared with the contribution of other contractors.

11.1 Planning

Our most signficant contribution is a formalization of the 'planning process'. Section 3 describes our mathematical model: it defines the 'plan design space' and the risk associated with a plan as a measure of performance, and presents an algorithm to find 'optimal plans'.

Importantly, the algorithm (called *Interactive Task Planner* or ITP) allows for 'variable autonomy', that is, a plan can be generated in a fully automatic manner, or it may be extensively modified by the planner in light of considerations that are not reflected in our 'risk' performance measure. The ITP offers a visual tool for the planner's intervention. The tool may be used by the planner to rapidly evaluate the changes in risks and flight paths resulting from proposed modifications in the plan. The tool produces a 'sensitivity table' that assists the planner in proposing changes.

The rapid evaluation of alternatives is made possible by a 'fast marching method' algorithm, which generates risk-minimizing paths. This assumes that the UAV dyamics are well-modeled by an 'isotropic' velocity field, so that the resulting value function satisfies the eikonal partial differential equation (24).

A very important implication of our model is that the risk in attacking a target depends on which targets have already been destroyed. As a consequence, one major component of a plan is its precedence relation (section 3.2), which restricts the sequence in which targets must be attacked. In terms of the ITP, this leads to organizing Blue's attack in 'waves', which the planner may modify.

A consequence of the precedence relation is that it leads in a reasonable way to the grouping of sub-tasks into tasks. In an automatic mode, these tasks are simply precedence 'chains' or 'sub-trees', as illustrated in the top of figure 14. However, the planner may group sub-tasks differently, as illustrated in the bottom of figure 14, based perhaps on similarity of target types or geographical location. This information is displayed in the ITP window (figure 5), facilitating the planner's exploration of the plan design space.

Once the tasks are defined, the 'configuration and schedule' module is invoked to determine the composition of the teams, one per task, and to refine the nominal path and precedence relation obtained from the ITP.

The composition of a team specifies the number and type of UAVs, and their configuration in terms of sensors and weapons. The calculation of a team composition relies on a table that gives the range of each weapon and its lethality (probability of target destruction). We have limited ourselves to the use of GPS bombs. We assume that a 95 percent probability of destruction is required, which in turn determines the number of weapons needed. Given the GPS bomb-carrying capacity, this yields the number of UAVs that are needed to execute a task. This number is 'inflated' to take into account the possibility that UAVs may themselves be destroyed.

The 'configuration and schedule' module further decomposes each sub-task into 'legs', refines the sub-task precdence relation to one on legs, and prescribes a path for each UAV in the team assigned to the sub-task. As each nominal path ends at a target, the path is divided into two legs: a 'safe' leg in which UAVs not attacking the target may stay, and an 'attack' leg in which the UAV is within the target's threat range. The attack leg is designed in such a way that the attack UAV flies with a heading directed at the target, thereby minimizing its signature.

11.2 Execution

There are two important contributions. The first is the two-level hierarchy of 'team coordinator' and 'UAV controller'. The latter in turn comprises several individual modules to manage way-point navigation, weapons, and sensors. Controllers at both levels can be expanded, as was discussed in section 5. Currently teams only carry out a strike task. But if other task types, such as search or jamming, are implemented, the team controller can be augmented in a modular fashion to accommodate these. Similarly, the vehicle controller can be augmented to include other maneuvers or tactics.

From a controller design viewpoint, the use of Shift as the design specificaiton language, in contrast to, say, Simulink, Matlab, or 'C', brings an enormous benefit in terms of conceptual unity and implementation. Being object-oriented, Shift permits controllers to be instantiated as needed. Shift offers high-level constructs, so that the *same* team controller can work with teams with a variable number of UAVs, and the team of UAVs can be dynamically reallocated during the execution phase. In principle, this can be done in any other programming language, but at a huge increase in complexity. Furthermore, although a Shift program can only be executed as a simulation, a very similar program written in commercial Teja language (which provides the same functionality as Shift) can also generate code for a variety of operating systems including Linux, VRTX, and QNX. Controllers designed in Teja can be tested in simulation mode, and then used to control hardware.

Lastly, because Shift has a strict semantics in terms of networks of hybrid automata, Shift controllers can, in principle, be verified. Although tools for verification cannot today work with general hybrid automata models, they can verify restricted models. Such verification could be invaluable.

11.3 State estimation

Our most signficant contribution here is a Bayesian model of knowledge about the 'battlespace' in terms of the probability distribution, P_{threat}, of the Red force. Since the threat depends on the set *Targets* of Red targets, this

is the probability distribution of a *set-valued* random variable. It is impossibly complex to deal in a quantitative manner with a general distribution of this kind.

By making an independence assumption, (32)-(33), we drastically reduce the memory needed to store the threat distribution. Moreover, as shown in sections 6.1 and 6.2, this assumption continues to hold after a strike or search task is completed. This makes it computationally feasible to design an online and recursive state estimation procedure. The mathematical procedure, $Post_{\mathit{threat}}$, is described in section 6. It is implemented in a Windows data base.

11.4 Re-planning

A major concern of MICA is to account for uncertainty. We have dealt with uncertainty in two ways. At the planning level, uncertainty is manifested in lack of full knowledge of the disposition of Red forces, modeled as a probability distribution, P_{threat}. The measure of risk along a path takes this distribution into account, see section 3.3.

It is much more difficult to deal adequately with uncertainty that arises during execution. The design of the controllers of teams and individual UAVs is predicated on certain assumptions: For example, an attack UAV expects its target to be at a certain location, and it expects the 'safe' leg to be free of threat.

However, these expectations may be violated. In principle, one may model these 'dis-expectations' probabilistically. Indeed, section 2.6 describes how uncertain threats might be treated probabilistically. But it is virtually impossible to anticpate and take into account all possible contingencies in such a probabilistic manner, especially in the execution phase, when UAVs are flying and time evolves. Moreover, such an approach quickly becomes computationally intractable (recall the 'curse of dimensionality').

Our novel contribution is to build *exceptions* into the controller structure. The controller declares an exception when events occur which make it impossible to execute the task in which it is currently engaged. For example, a UAV is unexpectedly destroyed so that the target which it was attacking now threatens the team. The controller 'rolls back' the team to what it considers a safe region, and calls the planner to intervene: in an automated mode the ITP is re-invoked, based on the new (and unanticipated) event that was encountered.

Lastly, an implicit assumption in the MICA program has been that teams are *static*—they are composed during the planning phase, and maintain their identity throughout the execution phase. In section 2.7, we show within a simplified model that allowing teams to be re-formed at the end of each wave can reduce the resource requirement. The resource savings will be larger the greater is the uncertainty in UAV attrition rates.

HICST's lasting contributions are: (1) a model for optimal plans and procedures to find them; (2) a model for the 'state of the world', and a state estimator; (3) a very concise and flexible structure for the team and UAV controllers, with a provision to generate exceptions. The controller, specified in Shift, can be used to assist in generating real-time code for hardware implementations. These contributions advance the attainment of MICA's objectives. Significant problems remain unsolved. These are discussed next.

12 Open problems

We discuss open problems using the same structure as in the previous section.

12.1 Planning

The limitations in our approach to planning can be classified in terms of the threat model (discussed in section 12.2), the performance measures, the sub-task structure, the fast marching method, and the treatment of uncertainty.

Threat model

Three limitations of the threat model need to be overcome. First, there is a need to consider mobile targets. This will require some models of mobility, which makes the state estimator computationally more complex.

Second, and conceptually more difficult, is the need to model 'integrated' defenses, in which the threat posed by targets whose radars function in a coordinated manner, is not the sum of the threats posed by each target, as in (20).

Third, and even more difficult, is the need to consider the threat from an intelligent adversary. The most straightforward model relies on game theory, which quickly leads to computationally intractable problems, which could be addressed by by resorting to approximations, either in terms of exploring Red moves or some higher-level abstractions. The latter is more promising, especially if those abstractions have intuitive meaning in terms of possible strategies that the adversary may be using.

Multiple performance measures

We have adopted a single measure, namely the risk along a path. However, other measures that reflect resource consumption, such as fuel, time, and weapons, may need to be included in the definition of the plan design space and the optimal plans. In principle, one can simply require that feasible plans defined in section 3.2 also satisfy constraints on resource consumed. However, this move complicates the search for an optimal plan, as indicated in section 2.8. Nonetheless, it is clear that the plan design space must explicitly include multiple performance measures.

Sub-task structure

We assumed that all sub-tasks have as their objective the destruction of a target. This is not enough. There are other important tasks such as 'search' that cannot be specified simply in terms of a target. If the objective of a search is to reduce the uncertainty about the Red force (see section 6.2), the consequence of executing a search may be to re-design the plan. The more significant searches are, the less meaningful it will be to separate 'planning' and 'execution' as we have done. The MICA structure separating these two phases is reasonable only in the context of very good prior information.

Fast marching method

The fast marching method is key to rapidly finding optimal paths. This speed makes it feasible for the planner to use the ITP as an *interactive* tool. However, the fast marching method assumes an isotropic velocity field. This means that the position of a UAV is only constrained by the inclusion,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \in B(x, y),$$

in which $B(x, y)$ is a circle of radius that can depend on location. (The 'circle' makes it isotropic.)

Extensions of the fast marching method can weaken the isotropic requirement. But taking full-fledged dynamics into account, such as in the form,

$$\dot{x} \in Ax + B\mathcal{U},$$

in which $\mathcal{U}$ is the control set, will require rather different computational methods.

Treatment of uncertainty

12.2 State estimation

A procedure that estimates the disposition of Red forces is essential. The procedure $Post_{threat}$ of section 6 is computationally feasible because of the independence assumption. It is easy to construct likely situations in which the available information makes the independence assumption untenable. A simple example is where the total number of targets in two areas is known, so that the number of targets in each area cannot be independent. Adding target mobility obviously complicates state estimation.

However, there are radically different formulations of threat that may be appropriate.

12.3 Execution

The controller architecture and the specific team, task, sub-task, and vehicle controllers presented in section 6 is a great advance over current practice in the specification of controller design. The specification takes advantage of the object-oriented nature of *Shift* and the abstract constructs available in *Shift*. In particular,

- Specification of controllers is separated from their instantiation;
- Controllers are hierarhically organized;
- Structure of multi-vehicle mission controllers is independent of the number of vehicles (because of the set construct in *Shift*);
- User intervention is explicitly made available at all levels of the hierarchy in terms of a well-defined interface of command and response messages;

- Controllers can be extended through specialization (because *Shift* allows inheritance.

As indicated, the controller design has well-specified provision by which a human opertor can intervene in planning and execution of the automated system. The outstanding open problem is to design rules according to which the automated system will 'ask' for human intervention. In most automated systems this is done through a system of 'alarms': the operator is signaled whenever some variables exceed a threshold. In MICA automation is carried to a much deeper level, and simple alarms are inappropriate. Rather, what is needed is a system of 'exceptions' that indicate the inability of the automated system to continue to fulfil its current tasks. A theory of exceptions needs to be developed.

12.4 Re-planning

Related to the last point above, is the open problem of designing automatic 'triggers' that call for re-planning.

References

[1] SHIFT website: www.path.berkeley.edu/shift

[2] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. AC-40, no. 9, pp. 1528–1538, 1995.

[3] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences, USA*, vol. 93, no. 4, pp. 1591–1595, 1996.

[4] D. Adalsteinsson and J. A. Sethian, "The fast construction of extension velocities in level set methods," *Journal of Computational Physics*, vol. 148, pp. 2–22, 1999.

[5] J.-C. Latombe, *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1991.

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[7] J. Barraquand, B. Langlois, and J. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[8] R. Kimmel and J. A. Sethian, "Optimal algorithm for shape from shading and path planning," *Journal of Mathematical Imaging and Vision*, vol. 14, no. 3, pp. 237–244, 2001.

[9] K. Konolige, "A gradient method for realtime robot control," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, (Takamatsu, Japan), pp. 639–646, 2000.

[10] A. Orda, "Routing with end-to-end QoS guarantees in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 365–374, June 1999.

[11] G. Liu and K. G. Ramakrishnan, "A*prune: An algorithm for finding k shortest paths subject to multiple constraints," in *INFOCOM 2001*, vol. 2, pp. 743–749, 2001.

[12] A. Puri and S. Tripakis, "Algorithms for routing with multiple constraints," in *AIPS 2002 Workshop on Planning and Scheduling using Multiple Criteria*, (Toulouse, France), pp. 7–14, April 2002.

[13] A. Sei and W. W. Symes, "Convergent finite-difference traveltime gradient for tomography," in *Proceedings of* 65^{th} *Society of Exploration Geophysicists Annual Meeting*, (Houston, TX), pp. 1258–1261, 1995.

[14] M. G. Crandall, L. C. Evans, and P.-L. Lions, "Some properties of viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American Mathematical Society*, vol. 282, no. 2, pp. 487–502, 1984.

[15] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik 1*, pp. 269–271, 1959.

[16] M. Falcone, "Numerical solution of dynamic programming equations," in *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman equations*, Birkhäuser, 1997. Appendix A of [23].

[17] H.-K. Zhao, "Fast sweeping method for Eikonal equations I: Distance function," tech. rep., UCI, Department of Mathematics, University of California, Irvine, CA, 92697-3875, 2002. Under review, SINUM.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. San Franscisco: McGraw-Hill, 1990.

[19] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. New York: Cambridge University Press, 1999.

[20] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences, USA*, vol. 95, no. 15, pp. 8431–8435, 1998.

[21] J. A. Sethian and A. Vladimirsky, "Ordered upwind methods for static Hamilton-Jacobi equations: Theory and algorithms," *SIAM Journal on Numerical Analysis*, vol. 41, no. 1, pp. 325–363, 2003.

[22] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao, "Fast sweeping methods for a class of Hamilton-Jacobi equations," *SIAM Journal on Numerical Analysis*, vol. 41, no. 2, pp. 673–694, 2003.

[23] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman equations*. Boston: Birkhäuser, 1997.

[24] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Minerva Report, http://iew3.technion.ac.il/Labs/Opt/*, 2003.

[25] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[26] G. Calafiore and M.C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *To appear in Mathematical Programming*, 2004.

[27] L. El Ghaoui and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM J. Optim.*, 9(1):33–52, 1998.

[28] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[29] M. Putterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley-Interscince, 1994.

[30] J. Filler and K. Vrieze, *Competitive Markov Decision Processes*. New York: Springer, 1996.

[31] D. Berstsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Massachusetts: Athena Scientific, 1996.

[32] H. Mine and S. Osaki, *Markov Decision Processes*. American Elsevier Publishing Company Inc, 1970.

[33] G. Siouris, *Optimal control and estimation theory*. New York, USA: Wiley-Interscience, 1995.

[34] S. Wilks, *Mathematical Statistics*. New York, USA: Wiley-Interscience, 1962.

[35] E. Lehmann and G. Casella, *Theory of point estimation*. New York, USA: Springer-Verlag, 1998.

[36] E. Lehmann, *Testing Statistical Hypothesis*. New York, USA: Wiley, 1986.

[37] J. Pitman, *Probability*. New York, USA: Springer-Verlag, 1993.

[38] H. Poor, *An introduction to signal detection and estimation*. New York: Springer-Verlag, 1988.

[39] E. Feinberg and A. Shwartz, *Handbook of Markov Decision Processes, Methods and Applications*. Boston: Kluwer's Academic Publishers, 2002.

[40] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, January, 2004.

[41] H. R. Schwarz and J. Waldvogel, *Numerical Analysis: A Comprehensive Introduction*. New York, USA: John Wiley and Sons, 1989.

[42] D. D. Farias and B. V. Roy, "The Linear Programming Approach to Approximate Dynamic Programming." submitted to Operations Research, 2002.

[43] L. G. Epstein and M. Schneider, "Recursive multiple-priors." to appear in Journal of Economic Theory, 2003.

[44] L. G. Epstein and M. Schneider, "Learning under ambiguity." http://www.econ.rochster.edu/Faculty/Epstein.html, 2002.

[45] G. Iyanger, "Robust dynamic programming." personal communication, 2003.

[46] G. H. Golub and C. F. Van Loan, "An analysis of the total least squares problem," vol. 17, pp. 883–893, 1980.

[47] A. Ng and M. Jordan, "Pegasus: A policy search method for large MDPs and POMDPs," in *the proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence*, 2000.

[48] A. Nilim, L. E. Ghaoui, M. Hansen, and V. Duong, "Trajectory-based Air Traffic Management (TB-ATM) under weather uncertainty," in *the proceeding of the 4th USA/EUROPE ATM R & D Seminar*, 2001.

[49] S. Kalyanasundaram, E.Chong, and N. Shroff, "Markov Decision Processes with uncertain Transition Rates: Sensitivity and robust control," tech. rep., Department of ECE, Purdue University, West Lafayette, Indiana, USA, March 2001.

[50] L. El-Ghaoui and A. Nilim, "Robust solution to the markov decision processes with uncertain transition matrices," Tech. Rep. UCB/ERL M02/31, Department of EECS, University of California, Berkeley, November 2002.

[51] A. Shapiro and A. J. Kleywegt, "Minimax analysis of stochastic problems," *Optimization Methods and Software*, 2002. to appear.

[52] A. S. Nowak, "On zero sum stochastic games with general state space. i," *Probability and Mathematical Statistics*, vol. 4, no. 1, pp. 13–32, 1984.

[53] C. C. White and H. K. Eldeib, "Markov Decision Processes with imprecise transition probabilities," *Operations Research*, vol. 42, no. 4, pp. 739–749, 1994.

[54] E. Altman and A. Hordijk, "Zero-sum markov games and worst-case optimal control of queueing systems," *QUESTA , a special issue on optimization of queueing systems*, vol. 21, pp. 415–447, 1994.

[55] Z. Chen and L. Epstein, "Markov Decision Processes with imprecise transition probabilities," *Econometrica*, vol. 70, pp. 1403–1443, 2002.

[56] M. Abbad and J. A. Filar, "Perturbation and stability theory for Markov control problems," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1415–1420, 1992.

[57] J. K. Satia and R. L. Lave, "Markov Decision Processes with Uncertain Transition Probabilities," *Operations Research*, vol. 21, no. 3, pp. 728–740, 1973.

[58] M. Abbad, J. Filar, and T. Bielecki, "Algorithms for singularly perturbed limiting average Markov control problems," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1421–1425, 1992.

[59] T. Ferguson, "Prior distributions on space of probability measures," *The Annal of Statistics*, vol. 2, no. 4, pp. 615–629, 1974.

[60] R. Givan, S. Leach, and T. Dean, "Bounded parameter Markov Decision Processes," in *Fourth European Conference on Planning*, pp. 234–246, 1997.

[61] J. Bagnell, A. Ng, and J. Schneider, "Solving uncertain Markov Decision Problems," Tech. Rep. CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon University, August 2001.

[62] A. Varma, "Case studies in joint estimation and optimization," Master's thesis, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 2001.

[63] J. K. Satia, *Markovian Decision Processes with Uncertain transition matrices or/and Probabilistic Observation of states*. PhD thesis, Department of Industrial Engineering , Stanford University, 1968.

[64] S. Boyd, "Convex Optimization," Tech. Rep. Course Reader, EECS 290N, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Fall 2001.

[65] L. El-Ghaoui, M. Oks, and A. Varma, "Constraint likelihood region of confidence of transition probability matrices in credit risk analysis," tech. rep., Department of EECS, University of California, Berkeley, USA, 2001.

[66] Y. Nesterov and A. Nemirovski, *Interior point polynomial methods in convex programming: Theory and applications*. Philadelphia, PA: SIAM, 1994.